Burroughs

B 1700 Systems Data Management System II (DMS II)

REFERENCE MANUAL

RELATIVE TO MARK V.0 RELEASE

PRICED ITEM

Printed in U.S.A.

January, 1976

1089794

Burroughs

B 1700 Systems Data Management System II (DMS II)

REFERENCE MANUAL

RELATIVE TO MARK V.0 RELEASE

PRICED ITEM

Printed in U.S.A. January, 1976 1089794

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Publications Department, Technical Information Organization, TIO-West, Burroughs Corporation, 9451 Telstar Avenue, El Monte, California 91731.

SECTION	TITLE	PAGE
	PREFACE	vi
	INTRODUCTION	vii
	B 1700 Data Management System II (DMSII) Components	vii
	Data Base Administrator	x
1	SYNTAX SPECIFICATIONS AND CONVENTIONS	1–1
	Syntax Specifications	1–1
	Syntax Conventions	1–2
2	DMSII STRUCTURE TYPES	2–1
	Introduction	2–1
	Data Set Set Structures	2–1
	Set and Subset Structures	2–1
	Automatic Sets	2–2
	Automatic Subsets	2–2
	Manual Subsets	2–2
	Structure Types	2–3
	Data Set With No Sets	2–3
	Data Set With Ordered Set	2–4
	Data Set With Embedded Data Set (No Sets)	2–5
	Data Set With Embedded Data Set And Ordering Set	2–6
	Data Set With No Ordering Set, Retrieval Set and Automatic Subset	2–8
	Data Set With Multiple Ordered Sets and One Retrieval Set	2–10
	Two Data Sets, One Referring To A Manual Subset of the other With No Key	2–12
	Two Data Sets Each Referencing A Subset Of The Other	2–14
3	DATA AND STRUCTURE DEFINITION LANGUAGE (DASDL)	3–1
	Data Base Description	3–1
	Data Base Syntax	3–1
	Disjoint Data Set	3–2
	VERIFY and REQUIRED ALL	3–2
	Semantics	3–3
	Record Description	3–5
	Data Item	3–5
	Group Item	3–6
	Embedded Data Set	3–7
	Embedded Set	3–8
	Subset	3–9
	Disjoint Set	3–11
	Key Structure	3–11
	Condition	3–12
	Physical Structures	3–14
	Disjoint Data Set Attributes	3–15
	Data Set Maintenance Techniques	3–16
	Index Sequential Attributes	3–17
	Index Random Attributes	3–20
	List Attributes	3–22
	File Attributes	3–23

SECTION	TITLE	PAGE
4	COBOL INTERFACE	4–1
	Introduction	4–1
	COBOL Data Division	4–4
	General	4–4
	DATA–BASE SECTION	4–4
	Data Set References	4–4
	Invoked Data Set	4–5
	Multiply-Invoked Data Set	4–6
	COBOL Procedure Division	4–6
	General	4–6
	Move And Move Corresponding	4–6
	Exception Processing	4–7
	ON EXCEPTION Clause	4–7
	DMSTATUS Register	4–8
	Selection Expression	44-10
	Key Condition	4–11
	Selection Expression Forms	4–11
	COBOL Statements	4–14
	CLOSE	4–15
	CREATE	4–16
	DELETE	4–17
	FIND	4–18
	FREE	4–19
	INSERT	4–20
	MODIFY	4–21
	OPEN	4–22
	RECREATE	4–23
	REMOVE	4–24
	STORE	4–25
Appendix A –	DMSII COMPILATION CUIDE	A–1
	DASDL Compilation Procedures	A–1
	DASDL Compilation Program Name	A–3
	DASDL File Names	A–3
	DASDL Compiler ($) Options	A–3
	COBOL Compilation Procedures	A–19
	Disk File Structures	A–19
	DMSII Operation and Console Printer Messages	A–21
	DMSII Error Messages	A–21
	Data Base Integrity	A–22
	Library Maintenance of DMSII Files	A–23
	Memory Dumps	A–23
Appendix B –	DASDL PHYSICAL STRUCTURES	B–1
	Disjoint Data Set Example	B–1
	Index Random Example	B–2
	Index Sequential Example	B–3
	Unordered List–embedded Data Set Example	B–4
	Unordered List – Subset Example	B–6
	Ordered List – Embedded Ordered Data Set Example	B–6
	Ordered List – Subset With A Key Example	B–7

SECTION	TITLE	PAGE
Appendix C –	DASDL GLOSSARY	C–1
Appendix D –	DASDL ERROR MESSAGES AND WARNINGS	D–1
	Introduction	D–1
	Warning Messages	D–1
	Error Messages	D–1
Appendix E –	COBOL EXTENSIONS	E–1
Appendix F – -	QUALIFYING A DATA BASE DESCRIPTION	F–1
Appendix G –	B 1700 – B 6700/B 7700 DMSII COMPATIBILITY	G–1
Appendix H –	DATA BASE EXAMPLE	H–1
	Introduction	H–2
	UNIV Data Base Identifiers	H–23

FIGURE	TITLE	
1	Simplified DMSII Compilation Process	viii
2	Simplified DMSII Object Program Execution	ix
4–1	Current–Record Pointer State	4–2
4–2	Set or Subset Current–Path Pointer State	4–3
A–1	Compile for Syntax	A–2
A–2	Compile for Library	A–2
A–3	UNIV Data Base Example	A–5
H–1	UNIV Data Base Diagram	H–2
H–2	DASDL Program Example	H–4
H–3	COBOL Program Example	H–13

TABLE	TITLE	
4–1	DMSTATUS Categories and Description	4–8

PREFACE

The B 1700 Data Management System II (DMSII) consists of the following three components: A Data And Structure Definition Language (DASDL) used to describe a data base, a COBOL interface providing programmatic access to the data in the data base, and the data access routines contained within the Master Control Program (MCP) that control data storage and retrieval. These three components form the nucleus of the B 1700 Data Management System II.

The information contained in this manual reflects System Software Release Mark V.0.

LIST OF APPLICABLE B 1700 PUBLICATIONS

Publication Title	Form Number
B 1700 System Software Operational Guide	1068731
B 1700 Systems COBOL Reference Manual	1057197

INTRODUCTION

B 1700 DATA MANAGEMENT SYSTEM II (DMSII) COMPONENTS

An overview of the B 1700 DMSII, illustrating the operational flow of DASDL, the COBOL interface, and the data access routines within the MCP, is shown in figures 1 and 2. The descriptions referenced in figures 1 and 2 are described below.

Reference	Description
A	A DASDL source deck defining the logical and physical specifications of the data base.
B	The DASDL compiler.
C	The DATABASE/LIBRARY created by the DASDL compilation provides the COBOL compiler with compilation information.
D	The DASDL dictionary file created by the DASDL compilation containing all the structural characteristics of the data base.
E	The data base files created by the DASDL compiler at the time the INITIALIZE statement is encountered.
F	The COBOL compiler.
G	The created object-code file.

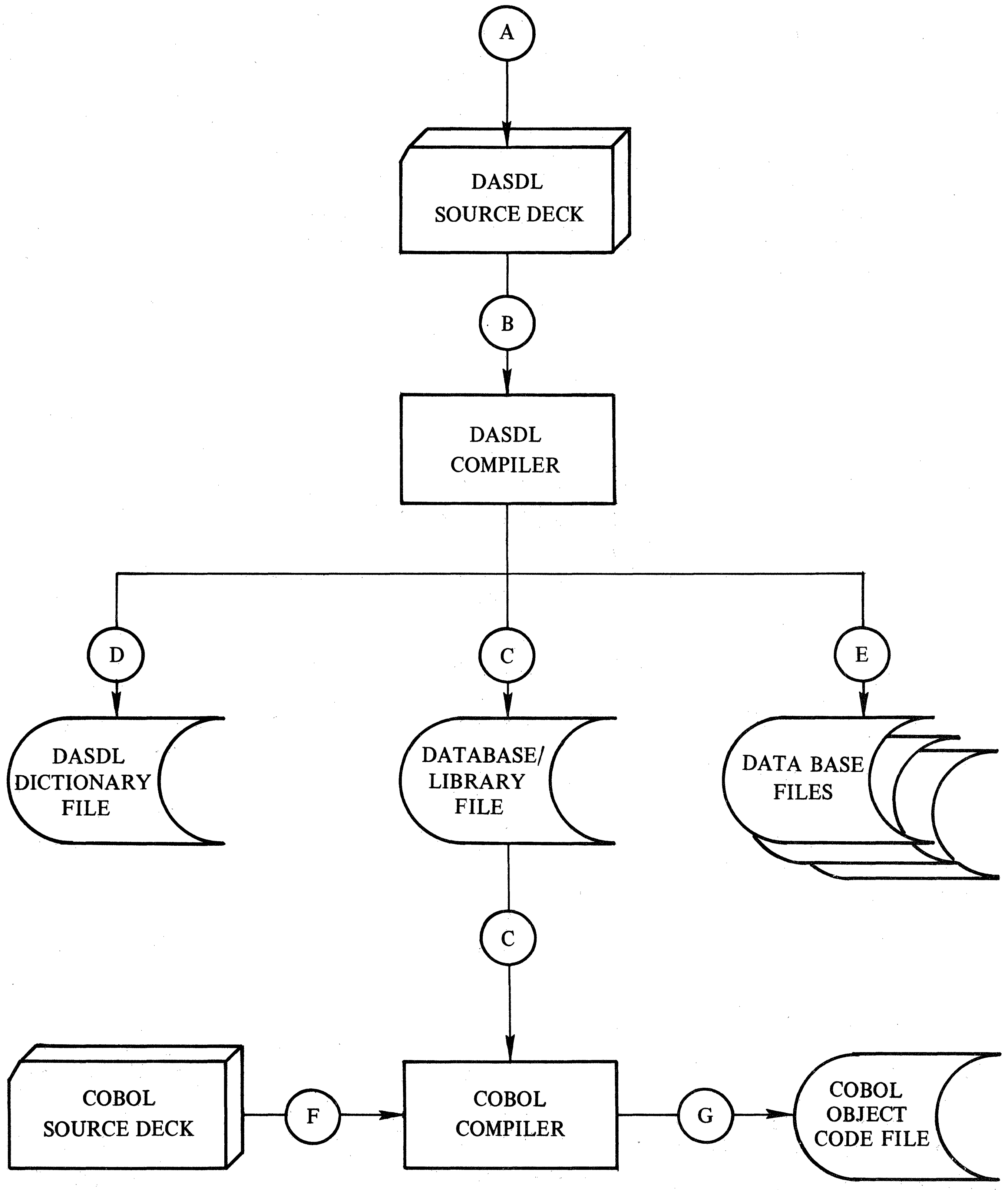


Figure 1. Simplified DMSII Compilation Process

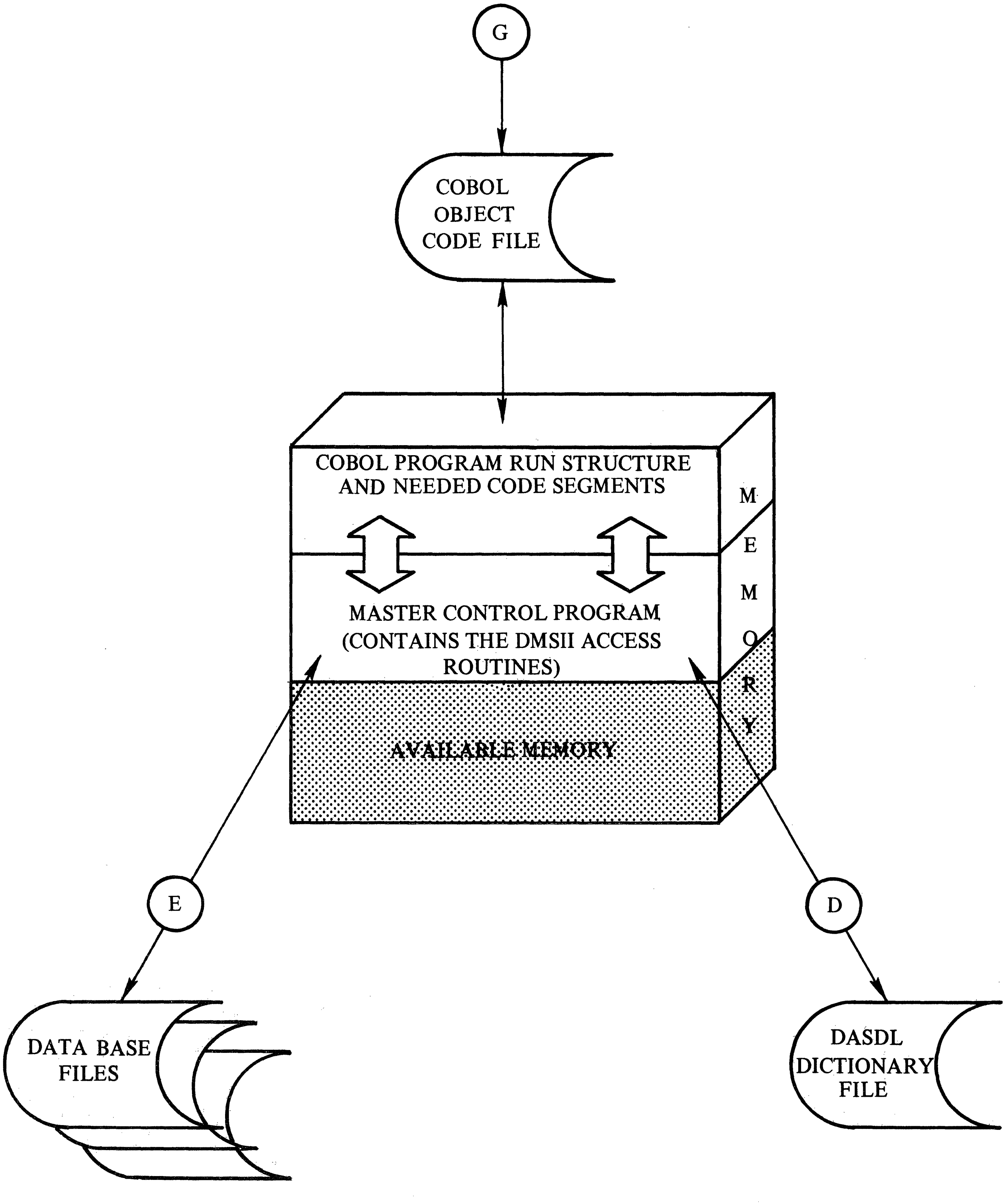

G
COBOL OBJECT CODE FILE
COBOL PROGRAM RUN STRUCTURE AND NEEDED CODE SEGMENTS
MASTER CONTROL PROGRAM (CONTAINS THE DMSII ACCESS ROUTINES)
AVAILABLE MEMORY
MEMORY
E
D
DATA BASE FILES
DASDL DICTIONARY FILE

Figure 2. Simplified DMSII Object Program Execution

DATA BASE ADMINISTRATOR

DASDL is the programmatic tool used by a person(s), usually referred to as a Data Base Administrator (DBA). It is one of the functions of the DBA to describe a data base to the B 1700 Data Management System II. The overall design of the data base is the responsibility of the DBA and includes the following:

a. Understanding the requirements of all users of the data base.

b. Analyzing the various demands to be made on the system.

c. Producing a data description capable of fulfilling the needs of all users of the system.

The DBA must also determine which applications require maximum optimization in order to provide for overall efficiency. Because DASDL allows the flexibility of many alternative solutions to a given problem, the DBA is always in a position to monitor and optimize the uses of the data base. The DBA must be aware of all factors and once the system is designed, must be committed to tailoring its structures.

Typically, the DBA produces a data base design by using the DASDL default options creating the data base structures. The DBA can then allow users to test the various applications. As experience is gained and the performance of the system is evaluated, the DBA can experiment with alternative solutions. The end result, therefore, reflects the decisions of the DBA in determining what is needed to produce the optimum usage of the data base for the entire organization rather than for any one application.

The types of decisions the DBA makes are based on evaluation of the critical resources. For example, at the cost of increasing memory used at program execution and increasing secondary storage space, the DBA may decide that some data should be stored in more than one location so all related information can be retrieved with one access. The DBA may also decide that the sequencing requirements of one application are used so rarely that an additional set to maintain that ordering is not worthwhile.

The DBA also evaluates the system requirements in terms of the structures and their physical parameters, depending on the needs of the installation. Initially, most questions relating to the physical parameters of the data base are less important than the logical structures required by the application programs. This makes the task of the DBA twofold:

a. Selecting structures based on their capabilities for supporting the logical requirements of the applications.

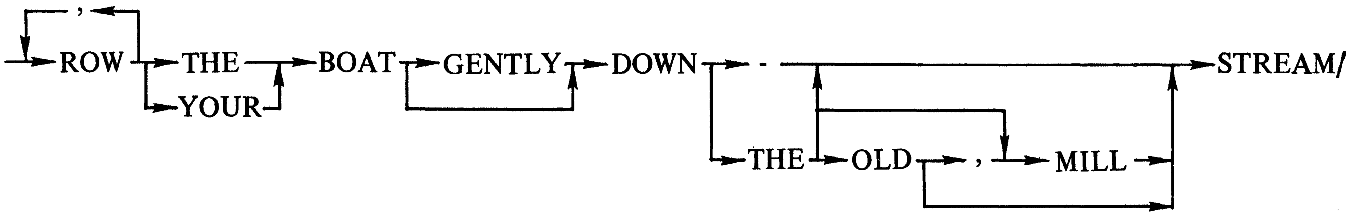
b. Optimizing the performance of the structures selected.

1. SYNTAX SPECIFICATIONS AND CONVENTIONS

SYNTAX SPECIFICATIONS

The principal means of displaying COBOL and DASDL statements is the syntax diagram. The syntax diagram technique affords a concise method of syntax illustration involving default options, alternatives, and iterations. The basic rule is that any path traced along the forward directions of the arrows produces a syntactically valid expression. The following examples illustrate the syntax diagram techniques.

Example 1:

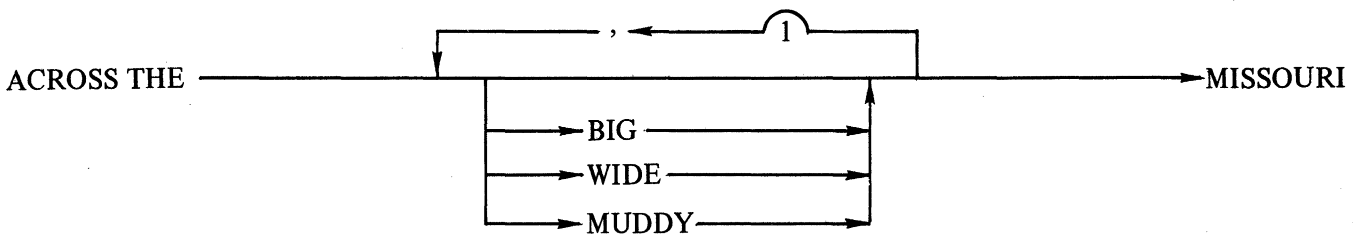


Valid statements from this example include:

ROW THE BOAT DOWN-STREAM.
ROW, ROW, ROW YOUR BOAT GENTLY DOWN THE STREAM.
ROW, ROW, ROW, ROW THE BOAT DOWN THE OLD STREAM.
ROW YOUR BOAT DOWN THE MILL STREAM.
ROW THE BOAT DOWN THE OLD, MILL STREAM.

Example 2:

The following convention is used to control the number of iterations:

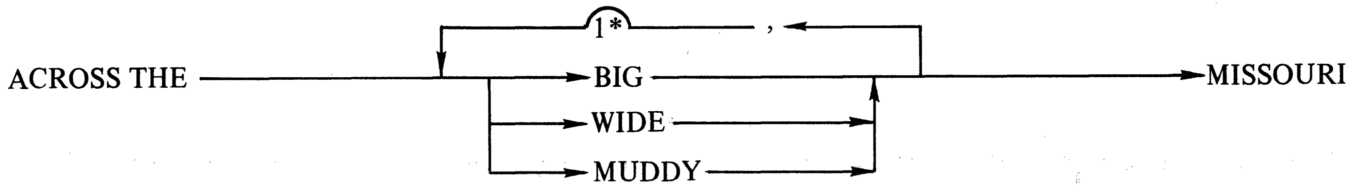


The bridge (⌒) over the "1" can be crossed only one time in forming a valid expression. Thus, a maximum of one comma and two adjectives can appear in a statement of this type. Valid expressions for this example include:

> **ACROSS THE MISSOURI.**
> **ACROSS THE WIDE MISSOURI.**
> **ACROSS THE BIG, MUDDY MISSOURI.**
> **ACROSS THE MUDDY, WIDE MISSOURI.**
> **ACROSS THE BIG, BIG MISSOURI.**

Example 3:

An asterisk (*) associated with a bridged number indicates that the path must be crossed at least one time. By changing example 2 to the following:



proper syntax is obtained by crossing the bridge at least one time. Valid expressions from this example include:

> **ACROSS THE BIG, BIG MISSOURI.**
> **ACROSS THE BIG, BIG, BIG MISSOURI.**
> **ACROSS THE BIG, WIDE, MUDDY MISSOURI.**

SYNTAX CONVENTIONS

DASDL and COBOL constructs consist of letters, digits, special characters, and blanks. Letters, digits, and hyphens are alphanumeric characters. All other non-blanks are delimiters. Alphanumeric characters can be aggregated into such syntactic items as integers, keywords, and identifiers. Keywords are reserved in DASDL and cannot be used in constructing identifiers. Rules and restrictions in construction of identifier names are identical to those for COBOL identifiers, except that no identifier can contain more than 17 characters (for compatibility with B 6700/B 7700 series systems DMSII) and all identifiers within a data base must be unique.

Keywords

All alphanumeric items appearing in capital letters are keywords and are used literally. Abbreviations are not allowed. Example: POPULATION

Blanks

Blanks separate syntactic items and can appear freely anywhere except within certain text fields, where they are significant characters. Blanks are optional on either side of a delimiter. Whenever one alphanumeric item, keyword, identifier, or integer, for example follows another with no intervening delimiter, it must be separated by at least one blank.

End-Of-Statement

An end-of-statement is indicated by an arrow followed by a slash. Example:

Syntactic Variables

All alphanumeric items that are not keywords in the syntax diagrams are syntactic variables, and represent information to be supplied by the user. A particular variable can represent a single character, a simple construct such as an integer or text string, or a complex construct. Most variables are defined programmatically where they are to be used.

BROKEN BRACKETS (   )

Left and right broken brackets containing letters, digits, or letters and digits represent a metalinguistic variable. When a metalinguistic variable appears in the text it is referring to its appearance in the syntax specification being discussed.

identifier

An identifier is a string of characters used to represent some entity, such as an item name composed of letters, digits, and hyphen. An identifier can vary in length from one through seventeen characters. The characters must be adjacent and the first character of an identifier must be a letter.

integer

An integer is specified by a string of adjacent digits representing the decimal value of the integer.

delimiter

A delimiter may generally be any non-alphanumeric character. The hyphen is excluded.

literal

A literal is a data item whose value is identical to the characters contained within the item.

PERCENT SIGN (%) (DASDL Only)

The percent sign is used to designate DASDL documentation or comments, and its presence terminates the scan of a card. The example below illustrates the usage of a percent sign for in-line coding.

```
00000100  :%THIS DASDL PROGRAM GIVES EXAMPLES
00000150  :%OF THE VARIOUS CONSTRUCTS USED IN
00000200  :%DASDL TO DESCRIBE A DATA BASE.
00000300  :PARAMETERS(
00000400  :         BUFFERS = 10   );
```

2. DMSII STRUCTURE TYPES

INTRODUCTION

A data base is constructed by a DASDL compilation. The contents and format of the data base are usually the responsibility of the Data Base Administrator (DBA). The DASDL compiler, using a description of the data base (DASDL source statements), produces a data base dictionary file containing information about each structure described within the data base.

Data base structures are either disjoint or embedded. A disjoint structure is free standing. A structure is considered embedded when it is declared as an item within some other structure. A structure can be one of three types: data set, set, or subset.

DATA SET STRUCTURES

A data set is similar to a conventional file in that it contains the actual records of information. However, it is different from a conventional file in that items within the record may themselves be structures, in which case, these items are considered as embedded structures. A record of a data set which contains an embedded structure is referred to as the owner record of the embedded structure. If the embedded structure is a data set, a record of the embedded data set is considered a detail record of the master. The DBA defines a data set, the items, and their attributes that form data set records, and also the physical organization of these records. The application programmer must be aware of these record items and attributes prior to accessing a data base. Knowledge of the physical organization of the data base is not required in order to access the data base.

Set and Subset Structures

Sets and subsets are structures organizing the records of a data set into some logical sequence. A set provides access to all of the records of a data set. A subset provides access to a limited collection of records of the data set. Since several sets or subsets can exist for the same data set, the same data can be accessed in several different sequences. For example, given a data set containing employee records, one set could order the data ascending order by the last name and another set could order the data in descending order by employee number. Those data items of a data record that are used to control the ordering of a set or subset are known as the key of the set or subset.

There are two methods of accessing a data set through a set or subset. The first method, accessing of records based on the value of key fields, is called the random access method. An example of the random access syntax is:

```
    FIND UNIV-COURSES VIA UNIV-C-SET AT CRS-NO  =  1234
```

The second method, accessing of records sequentially based on the value of the key fields, is the serial access method. An example of the serial access syntax is:

```
    FIND UNIV-COURSES VIA NEXT UNIV-C-SET
```

Records may also be accessed based on the physical ordering of the records within the data set. The physical ordering may or may not correspond to the order in which the records were created. An example of access based on the physical ordering of a data set is:

```
FIND NEXT UNIV-COURSES
```

Automatic Sets

All sets are automatic in that as new records are stored, the system automatically creates entries in the set for those new records of the data set. Deleting records from a data set also automatically removes the entry from the set. Sets can be either embedded or disjoint structures.

Automatic Subsets

Subsets can be manual or automatic. Automatic subsets specify a condition for membership in the subset; the condition is checked each time a record is to be added to the data set. If the condition is met, the system automatically creates an entry in the subset. Those data records that meet the condition can be accessed by the automatic subset. Deleting a record from the data set removes the entry from the automatic subset if the subset entry exists. During an update, the condition is checked and the subset entry can be created or deleted. Automatic subsets can be disjoint structures only.

Manual Subsets

A manual subset requires the application program to insert the record in the manual subset after creating and storing a record in a data set. This requirement establishes an entry in the manual subset for the record in the data set. When deleting a record, it is necessary for the application program to remove the entry from the manual subset prior to deleting the record from the data set. Manual subsets can be embedded structures only.

STRUCTURE TYPES

Some examples of the structure types that form a data base are illustrated in the following text.

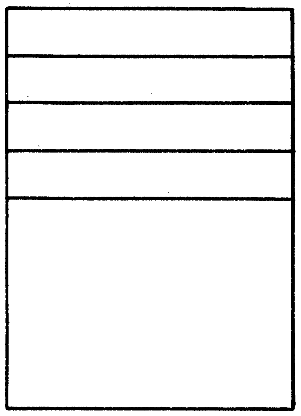
Data Set With No Sets

A data set with no sets might be illustrated using a payroll application, in which every record in the data set is accessed during the processing of the payroll program.

Coding Example:

```
PAYROLL DATA SET
  (.
   . (data set items)
   .
  ), POPULATION = 1000;
```

Physical Structure:



Record Access:

a. New records are stored in the first available location.

b. The records can be accessed based on the physical ordering of the data set. For example:

   FIND FIRST, FIND NEXT . . .

c. Records cannot be accessed based on data values.

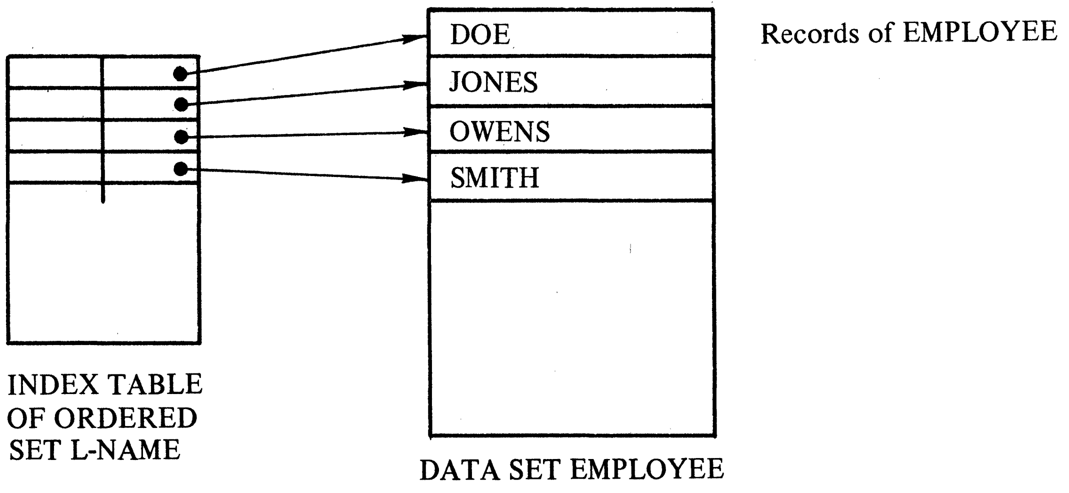
Data Set With Ordered Set

A data set with an ordered set could be used for an employee file with the last name as the key. The entire data set could be accessed through the set in alphabetical order by using the last name as the key, or any individual record could be accessed by using the last name of the individual as the key.

Coding Example:

```
EMPLOYEE DATA SET
  (LAST-NAME . . .
   .
   .
   .
  ),POPULATION = 1000;
   L-NAME ORDERED SET OF EMPLOYEE (LAST-NAME);
```

Physical Structure:



Record Access:

a. Records can be accessed based on the physical ordering of the data set. For example:

```
FIND NEXT EMPLOYEE
```

b. Records can be accessed based on the ordering sequence of the set. For example:

```
FIND EMPLOYEE VIA NEXT L-NAME
```

c. Records can be accessed based on the data value of a key. For example:

```
FIND EMPLOYEE VIA L-NAME AT LAST-NAME = "JONES"
```

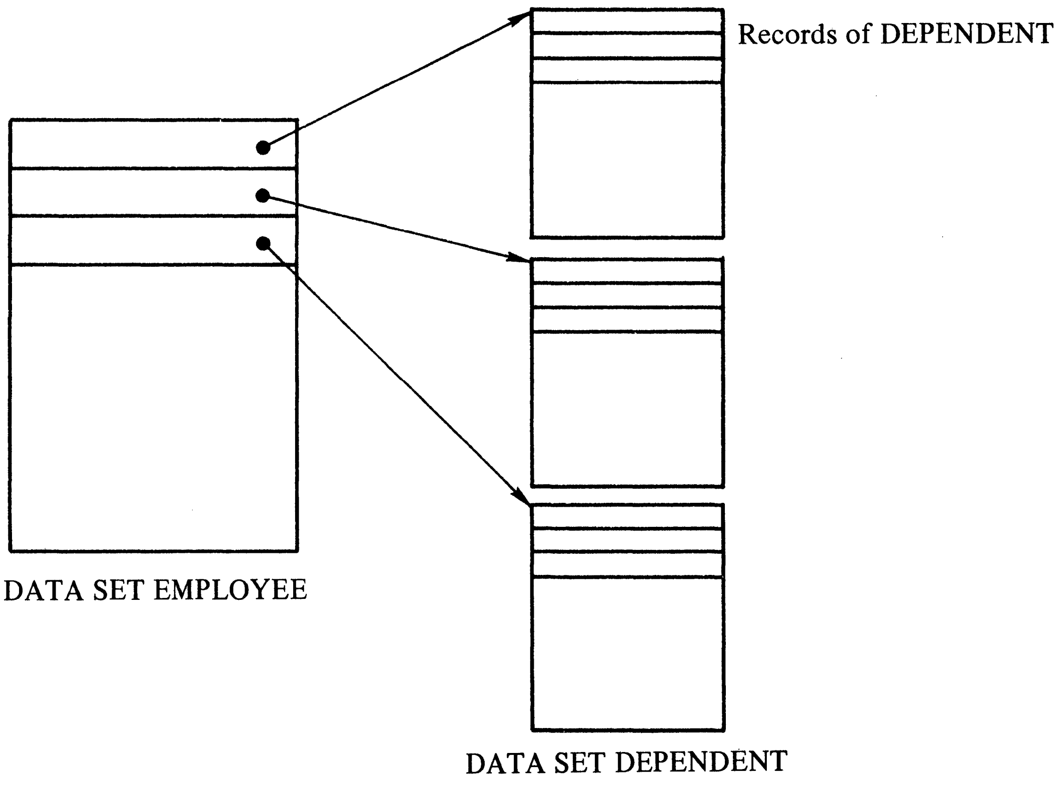
Data Set With Embedded Data Set (No Sets)

A data set with an embedded data set could be used for an employee file in which an embedded data set was used to account for each of the employee's dependents.

Coding Example:

```
    EMPLOYEE DATA SET
      (.
       .
       .
       DEPENDENT UNORDERED DATA SET
        (.
         .
         .
        ),POPULATION = 10
      ),POPULATION = 1000;
```

Physical Structure:



Record Access:

a. Records of data set DEPENDENT can be accessed based on the physical ordering of the embedded data set. For example:

   FIND NEXT DEPENDENT

b. There must be a valid EMPLOYEE current record in order to access a DEPENDENT record.

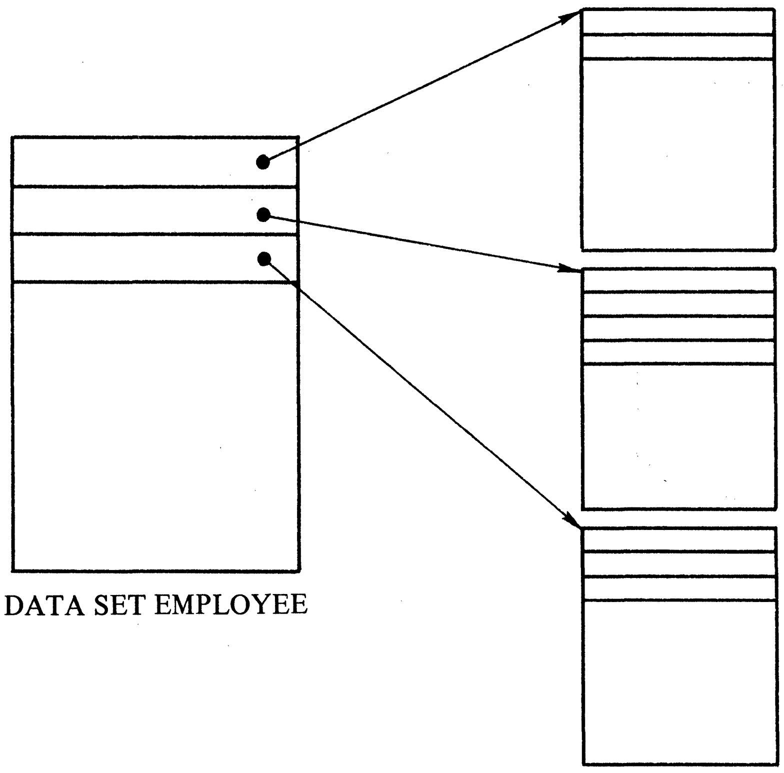
Data Set With Embedded Data Set And Ordering Set

This data structure could be used with the employee file as the data set and the employee job history as the embedded data set ordered by the job position.

Coding Example:

```
EMPLOYEE DATA SET
  (.
   .
   .
   JOB-HISTORY ORDERED DATA SET
     (POSITION ALPHA (20)
       .
       .
       .
     ), POPULATION = 10;
       JOB-POSITION ACCESS TO JOB-HISTORY KEY  (POSITION)
  ), POPULATION = 1000;
```

Physical Structure:



DATA SET EMPLOYEE

DATA SET JOB-HISTORY

Record Access:

a. Records of data set JOB-HISTORY can be accessed based on the ordering sequence of JOB-POSITION. For example:

```
FIND JOB-HISTORY VIA NEXT JOB-POSITION
```

b. Records can be accessed based on the data values of the key. For example:

```
FIND JOB-HISTORY VIA JOB-POSITION AT POSITION = SYSTEMS-ANALYST
```

c. There must be a valid EMPLOYEE current record to access any JOB-HISTORY record.

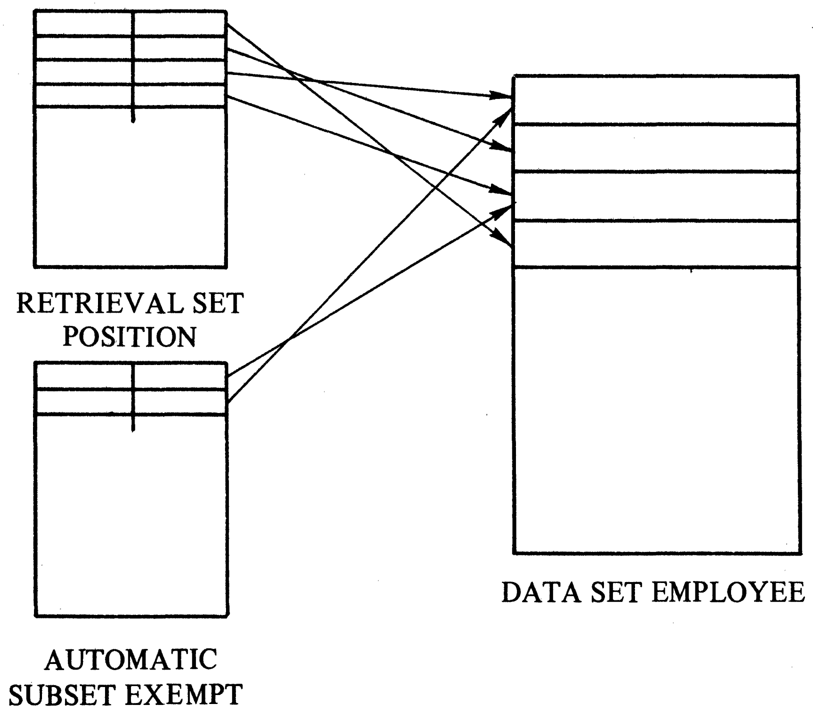
Data Set With No Ordering Set, Retrieval Set, and Automatic Subset

A data set with a retrieval set could be used with an employee file so that given a title and department the record for the employee who holds that position could be accessed. An automatic subset provides access to all the records of exempt employees.

Coding Example:

```
EMPLOYEE DATA SET
  (TITLE . . .
   DEPARTMENT . . .
   EXEMPT-STATUS
   NAME
   .
   .
  ), POPULATION = 1000;
   POSITION RETRIEVAL SET OF EMPLOYEE KEY (TITLE,DEPARTMENT) DUPLICATES;
   EXEMPT SUBSET OF EMPLOYEE WHERE (EXEMPT-STATUS = 1) KEY IS (NAME) ;
```

Physical Structure:



Record Access:

a. Records can be accessed based on the physical ordering of the data set. For example:

   FIND NEXT EMPLOYEE

b. Records can be accessed based on the value of a retrieval key. For example:

   FIND EMPLOYEE VIA POSITION AT TITLE = SECRETARY AND DEPARTMENT = SYSTEMS-PROGRAMMING

c. Records that satisfy the automatic subset condition can be accessed based on the physical ordering of the automatic subset. For example:

```
FIND EMPLOYEE VIA NEXT EXEMPT
```

d. Records that satisfy the automatic subset condition can be accessed based on the value of the subset key. For example:

```
FIND EMPLOYEE VIA EXEMPT AT NAME = "JOE DOE"
```

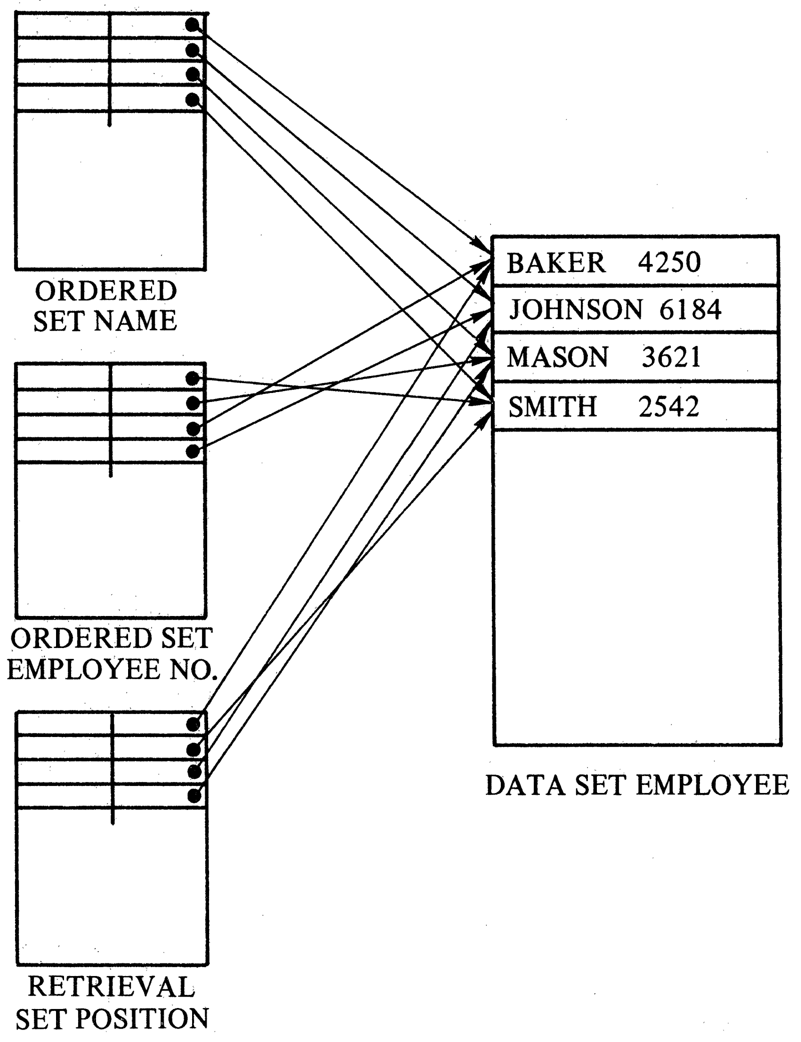
Data Set With Multiple Ordered Sets and One Retrieval Set

This data set could be an employee file ordered by name and employee number and retrieved by title and department.

Coding Example:

```
EMPLOYEE DATA SET
   (FIRSTNAME . . .
    LASTNAME . . .
    EMPLOYEE-NO . . .
    TITLE . . .
    DEPARTMENT . . .
    .
    .
   ),POPULATION = 1000;
    NAME ORDERED SET OF EMPLOYEE KEY (LASTNAME,FIRSTNAME);
    EMP-NO ORDERED SET OF EMPLOYEE KEY (EMPLOYEE-NO);
    POSITION RETRIEVAL SET OF EMPLOYEE KEY (TITLE,DEPARTMENT);
```

Physical Structure:



Record Access:

a. Records can be accessed based on the physical ordering of the data set. For example:

```
FIND NEXT EMPLOYEE
```

b. Records can be accessed based on any ordering sequence. For example:

```
FIND EMPLOYEE VIA NEXT EMPLOYEE-NO
```

The order however, is based on the values within the records, not the physical order of the records.

c. Records can be accessed based on data values of the order key. For example:

```
FIND EMPLOYEE VIA NAME AT LASTNAME = "SMITH" AND
FIRSTNAME = "JOHN"
```

d. Records can be accessed based on data value of a retrieval key. For example:

```
FIND EMPLOYEE VIA POSITION AT TITLE = MANAGER
AND DEPARTMENT = FINANCE
```

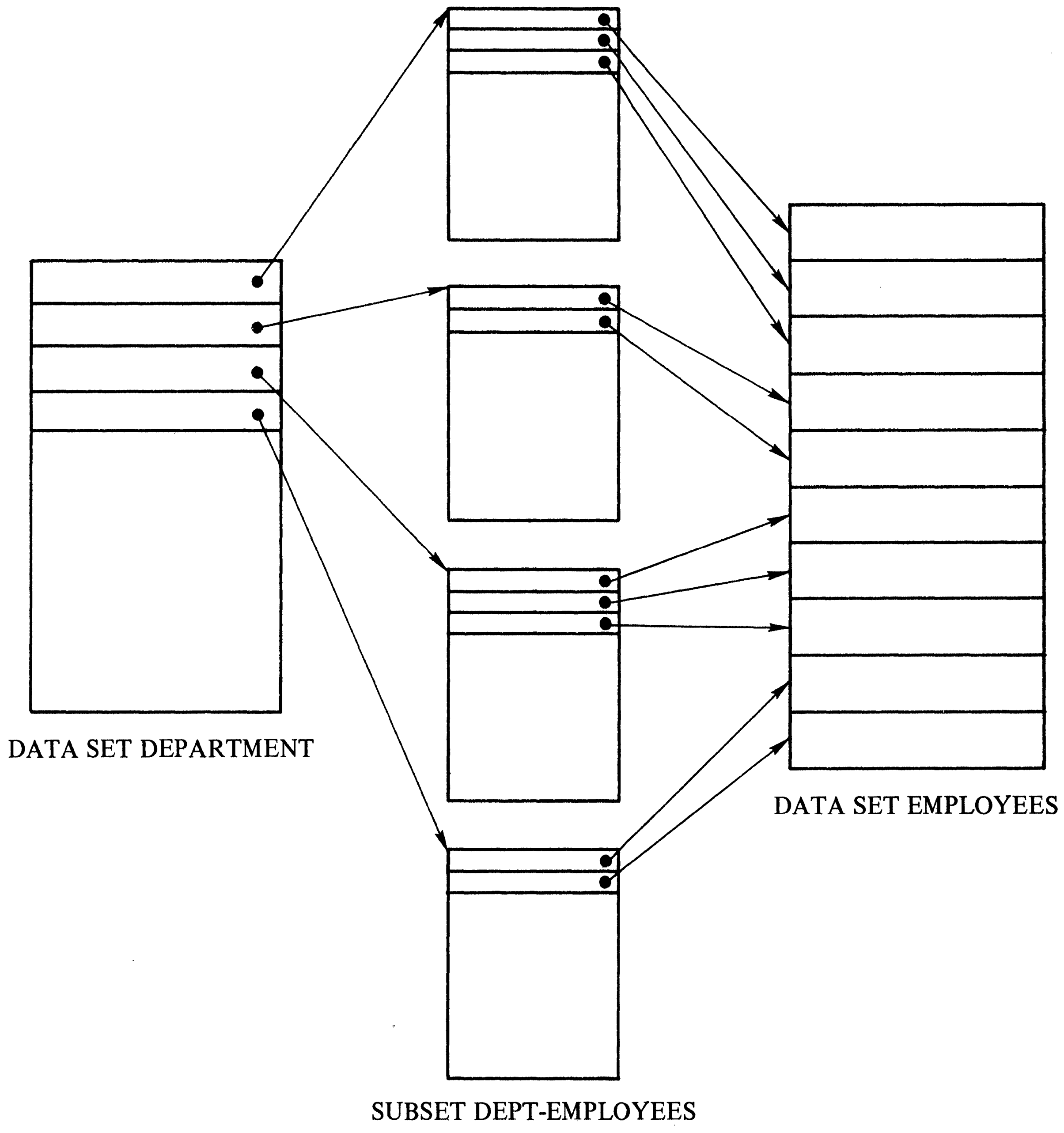
Two Data Sets, One Referring To A Manual Subset of The Other With No Key

This data structure could represent the relationship between departments and employees, with each department having a manual subset referencing all the employees of that department.

Coding Example:

```
DEPARTMENT DATA SET
  (.
   .
   .
   DEPT-EMPLOYEES SUBSET OF EMPLOYEES
   .
   .
  ), POPULATION = 10;
EMPLOYEES DATA SET
  (.
   .
   .
  ), POPULATION = 1000;
```

Physical Structure:



Record Access:

a. Records of data set EMPLOYEES can be accessed based on the physical ordering of a subset for a data set: For example:

   FIND EMPLOYEES VIA NEXT DEPT-EMPLOYEES

b. Records of data set EMPLOYEES can be accessed by the physical ordering of the data set. For example:

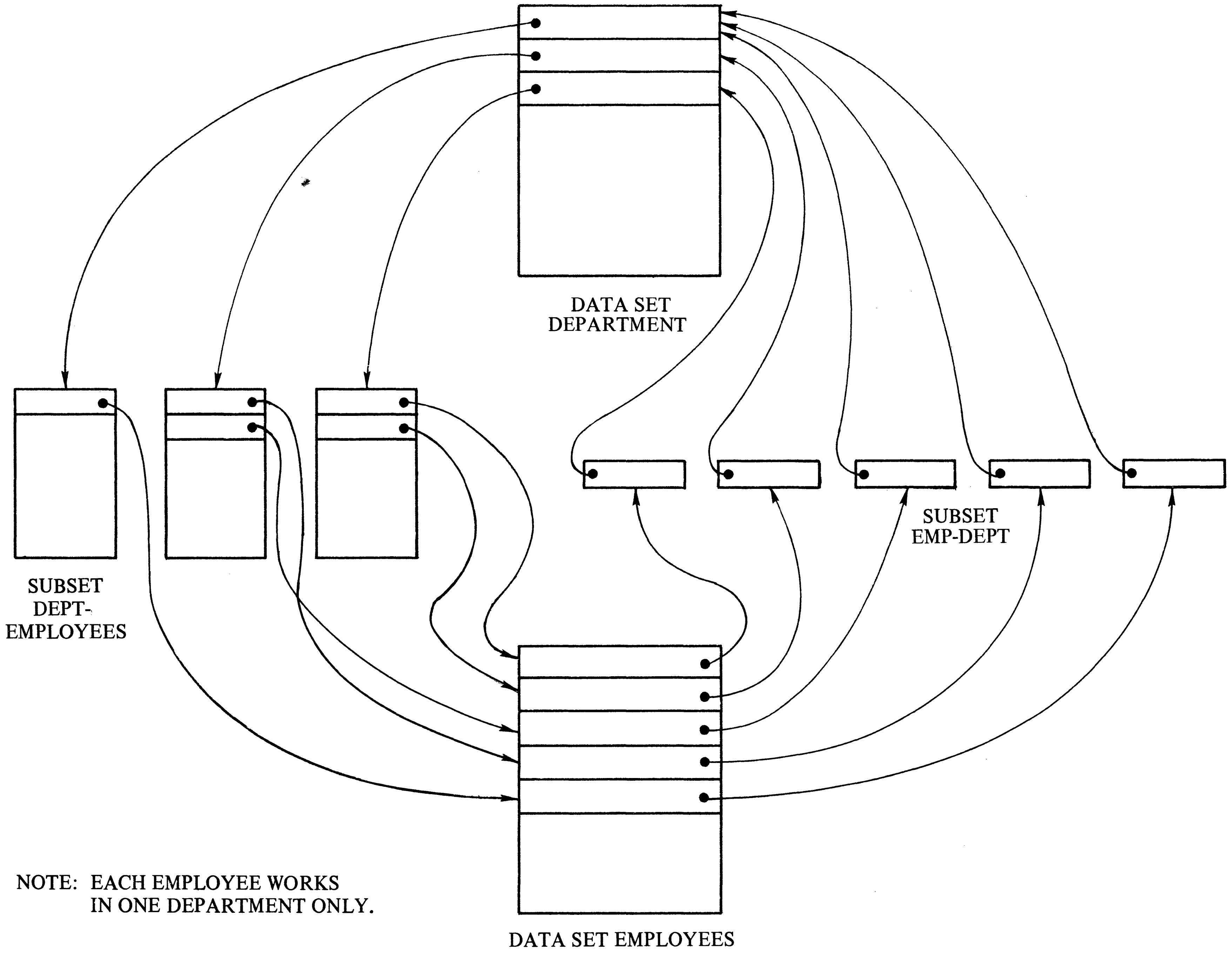
   FIND NEXT DEPARTMENT

Two Data Sets Each Referencing A Subset Of The Other

The preceding example could be expanded to order the employees within a department by their last name. Also there could be a manual subset within each record of data set EMPLOYEES referencing the department in which the employee works.

Coding Example:

```
DEPARTMENT DATA SET
  (.
   .
   .
   DEPT-EMPLOYEES SUBSET OF EMPLOYEES KEY (LASTNAME)
  ), POPULATION = 10;
EMPLOYEES DATA SET
  (LASTNAME . . .
   .
   .
   EMP-DEPT SUBSET OF DEPARTMENT
  ), POPULATION = 1000;
```



NOTE: EACH EMPLOYEE WORKS
IN ONE DEPARTMENT ONLY.

Record Access:

a. The records of data set EMPLOYEES can be accessed based on the physical ordering of a subset of a data set. For example:

   FIND EMPLOYEES VIA NEXT DEPT-EMPLOYEES

b. The records of data set DEPARTMENT can be accessed based on the data value of a ordered key of the subset. For example:

   FIND EMPLOYEES VIA DEPT-EMPLOYEES AT LASTNAME = "JONES"

c. Records of data set DEPARTMENT can be accessed based on the physical ordering of the data set. For example:

   FIND FIRST DEPARTMENT

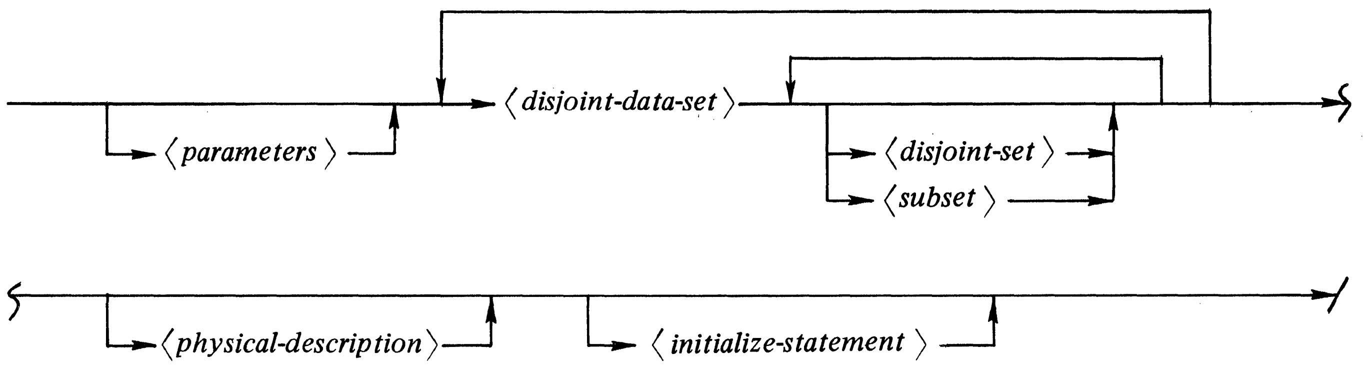
d. A master data set must have a current record to access its subset.

3. DATA AND STRUCTURE DEFINITION LANGUAGE (DASDL)

DATA BASE DESCRIPTION

A data base is described to the system by means of a DASDL compilation. Data sets, sets, and subsets constitute the data base. Appendix A contains specific information on the DASDL compilation procedure. This section describes the components and the structuring of a data base.

Data Base Syntax



Parameters:

——→ PARAMETERS ——→ (BUFFERS = ⟨*integer*⟩) ; ——→/

Initialize Statement:

——→ INITIALIZE ; ——→/

Semantics:

a. Data sets and sets on the outermost level of the description are disjoint data sets, and disjoint sets. A data base must contain at least one disjoint data set.

b. A data set description provides for specification of the logical structure of a file.

c. Set and subset descriptions provide logical specifications of indexes or index tables (paths) that are used in storage and retrieval of data contained in a data set.

d. The BUFFERS statement defines the optimum number of data buffers the system should try to utilize. The system adjusts dynamically to handle peak volumes of DMSII operations. By default, the system will utilize 10 buffers. The range can be from 3 to 100.

e. The INITIALIZE statement causes the DMSII files to be initialized by DASDL, and must be performed before there can be any access to a data base. INITIALIZE must be the last statement in the DASDL source deck.

f. Subset and set specifications cannot precede the specification of the data set that they reference.

DISJOINT DATA SET

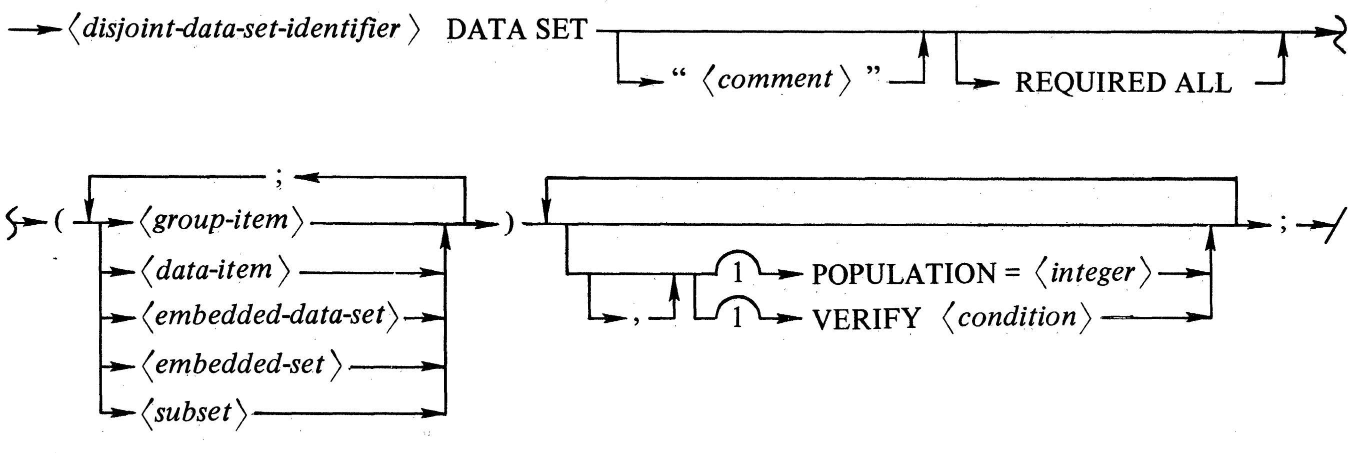
A data set consists of a collection of records that constitute a data set. All records in a data set are structured alike, with only the values of the data items differing. Describing the data set is accomplished by describing the data items within the records of the data set.

The data structure in a data set definition are represented in a COBOL-like format. Disjoint data sets appear on the outermost level of the description and are referred to as level 1. All data items contained in a data set are assigned a level that is one greater than that of the data set itself. All items of a group item are assigned a level that is one greater than that of the group item itself. DASDL represents data structures by the use of parentheses, with each set of parentheses representing a level.

VERIFY and REQUIRED ALL

The VERIFY condition and the REQUIRED ALL statement provide the mechanism for specifying the minimum criteria that a record must meet prior to its being stored in a data base. These specifications are checked for each record to be stored. The REQUIRED ALL statement specifies that all applicable items must be assigned a non-null value (where "null" is defined as all bits on). The VERIFY condition provides the specifications for complex comparisons and operations in determining the validity of a data record.

Syntax



SEMANTICS

a. DASDL recognizes five item types as valid components of disjoint data set record descriptions:

   1. Embedded data set.

   2. Embedded set.

   3. Subset.

   4. Group item.

   5. Data item.

   This record description of a disjoint data set specifies the format of a record of a data set. These records are stored in a physical file.

b. The record of a data set containing an embedded data set is referred to as being the "owner" of the records of the embedded data set.

c. The records of an embedded data set are referred to as members of its disjoint data set.

d. A disjoint set relates only to a disjoint data set. An embedded set and the embedded data set to which it relates must be on the same level description.

e. Files for the data sets and tables built for sets, if there are keys, use the POPULATION statement as a guide for area size for files and table size unless other specifications are given in the physical description.

f. The quoted comment ("comment . . ."), having a limit of 172 characters, provides a facility for inclusion of descriptive information in the data set. Continuation of quoted character strings across card boundaries requires a quote at the beginning of subsequent cards.

g. The REQUIRED ALL statement, if present, is equivalent to the REQUIRED statement on all items of the data set in that for each data item defined in the data set a value must be present for the record to be stored.

h. The VERIFY condition specifies a certain condition to be satisfied by the items of a record in order for it to be stored in a data set. If the condition is not met, the record will not be stored.

Example:

The following example contains the usage of a disjoint data set.

```
00005400  :MSF  DATA  SET(                    DISJOINT DATA SET
00005500  :     SSNO NUMBER(9);
00005600  :     NONAM NUMBER(1);
00005700  :     LNAME ALPHA(30);                 DATA ITEMS
00005800  :     MNAME ALPHA(30);
00005900  :     FNAME ALPHA(30);
00006000  :     CAMPUS-ADDRESS GROUP(            GROUP ITEM
00006100  :             DORM ALPHA(6);
00006200  :             ROOM NUMBER(4);          DATA ITEMS
00006300  :             POROX NUMBER(4);
00006400  :             PHONE NUMBER(7));
00006500  :     ND NUMBER(2);
00006600  :     DEGREE ALPHA(4) OCCURS 6 TIMES;
00006700  :     TOTHRS NUMBER(3);
00006800  :     TOTQP NUMBER(3);
00006900  :     GRADE-POINT-AVG NUMBER(3,2);
00007000  :     MJR NUMBER(3);
00007100  :     AMJR ALPHA(18);
00007200  :     SSEX NUMBER(1);
00007300  :     SAGE NUMBER(2);
00007400  :     HOME-ADDRESS SUBSET OF ADR;          MANUAL SUBSET
00007500  :     QUARTER      ORDERED DATA SET(
00007600  :             QTR    ALPHA(4);
00007700  :             QTTRHRS NUMBER(2);
00007800  :             QTRQP NUMBER(2);
00007900  :             CORSES ORDERED DATA SET(
00008000  :                   TYPECOURSE NUMBER(1);
00008100  :                   YR NUMBER(2);
00008200  :                   Q NUMBER(2);                      EMBEDDED
00008300  :                   GCRS SUBSET OF UNIV-COURSES;      DATA SET
00008400  :                   GGD ALPHA(2);
00008500  :                   TITLE-OF-PAPER ALPHA(30);
00008600  :                   PPRGD ALPHA(2));
00008700  :                   POPULATION  =  4;
00008800  :                   CSET ACCESS TO CORSES KEY IS      EMBEDDED
00008850  :                       (TYPECOURSE) DUPLICATES)      SET
00009000  :             POPULATION  =  5000;
00009100  :             QSEY ACCESS TO QUARTER KEY IS (QTR));
00009200  :     MSFSET ORDERED SET OF MSF KEY IS (SSNO);         DISJOINT SET
```

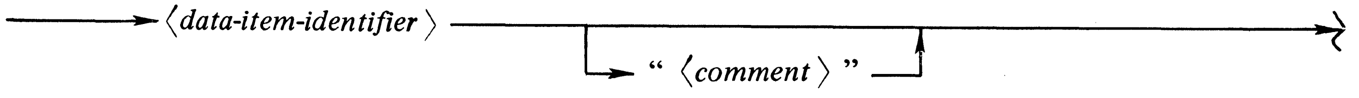
RECORD DESCRIPTION

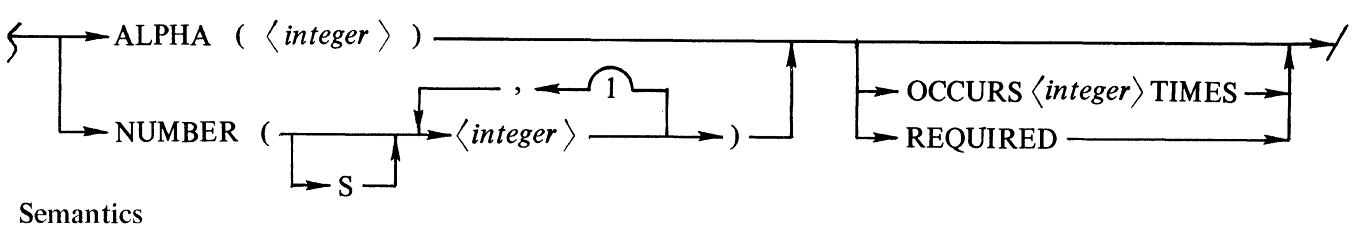
The five item types comprising the record description of a disjoint data set are the embedded data set, embedded set, subset, group item, and the data item. All items comprising the record description are separated by semicolons.

Data Item

Each data item in the record is described by an identifier, an optional description, and its data type. A data item can also have an occurrence (OCCURS) specification or a REQUIRED specification requiring the data item to be assigned a non-null value prior to its storage in the data set.

Syntax





Semantics

a. Comments enclosed within quotation marks are used for documentation and are stored in the dictionary file, but comments following the percent sign (%) are not stored.

b. The ALPHA specification handles strings of alphabetic characters, special characters, or digits. The size of the data item (the number of characters the data item can hold) is specified by an integer, enclosed within parentheses.

c. The NUMBER specification handles signed and unsigned numeric fields, either decimal or integer. The maximum size of a NUMBER statement is 23 unsigned digits or 22 signed digits when not a KEY item; and 12 unsigned digits or 11 signed digits when NUMBER is a KEY item.

d. When two integers are used to specify the size of a NUMBER item, the integer to the left of the comma specifies the total field width; the integer to the right of the comma is the number of digits after an implied decimal point.

e. The OCCURS clause of DASDL is identical to the COBOL OCCURS clause. The item must be subscripted when used. A maximum of three levels of subscripting is allowed. The number of occurrences is limited to 1023.

f. The maximum record size is 8192 characters.

g. Required items must be present and be non-null to be stored. The REQUIRED ALL option for a data set makes all items REQUIRED except those for which the REQUIRED statement is an invalid option.

h. The REQUIRED statement cannot be specified for any items appearing within the scope of an OCCURS clause. The REQUIRED ALL statement of a data set does not make an occurring type item required.

i. COBOL requires that all ALPHA and GROUP items start on byte boundaries. In order to satisfy this requirement, one-digit fillers are inserted, where necessary, with a warning message given.

Example:

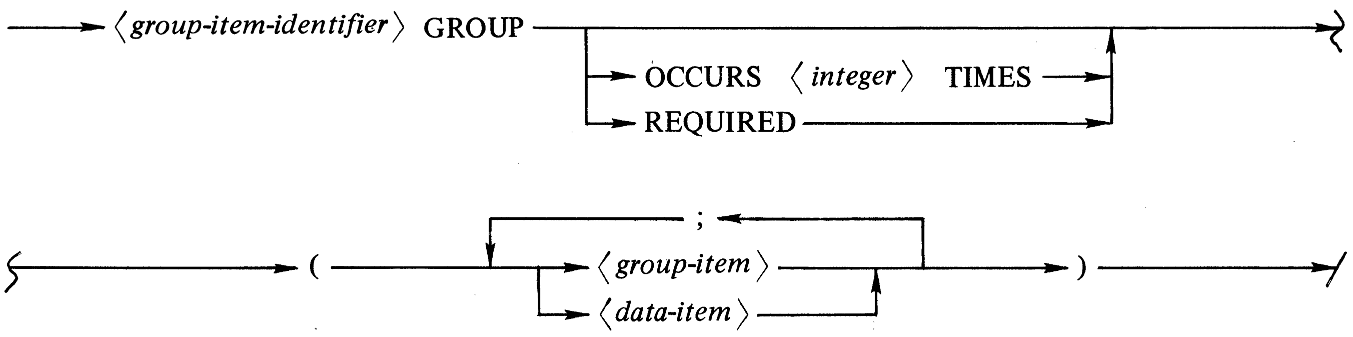
The following example illustrates a record description.

```
00006500  :      ND_NUMBER(2) ;
00006600  :      DEGREE ALPHA(4) OCCURS 6 TIMES;
00006700  :      TOTHRS NUMBER(3) ;
00005800  :      TOTQP NUMBER(3) ;
00006900  :      GRADE-POINT-AVG NUMBER(3,2) ;
00007000  :      MJR NUMBER(3) ;
00007100  :      AMJR ALPHA(18) ;
00007200  :      SSEX NUMBER(1) ;
00007300  :      SAGE NUMBER(2) ;
```

Group Item

Group items are used to establish hierarchical relationships within one record in the same manner that COBOL uses level numbers. Each group item in the record is described by an identifier followed by the word GROUP, an optional OCCURS or REQUIRED clause, and a list of data items or group items in any combination.

Syntax



Semantics

a. Group items are items that themselves contain items. Items within a group are declared at a level that is one greater than the level of the group.

b. Items that belong to groups are restricted to data items and further group items. Data sets, sets, and subsets are not allowed as items within a group.

c. The optional OCCURS clause can be nested to three levels. Each occurrence has a limit of 1023 times.

d. Group items can be REQUIRED if the group item does not appear within the scope of an OCCURS clause. The effect of this is to make all items within the GROUP required (REQUIRED) except those items for which the option is invalid.

Example:

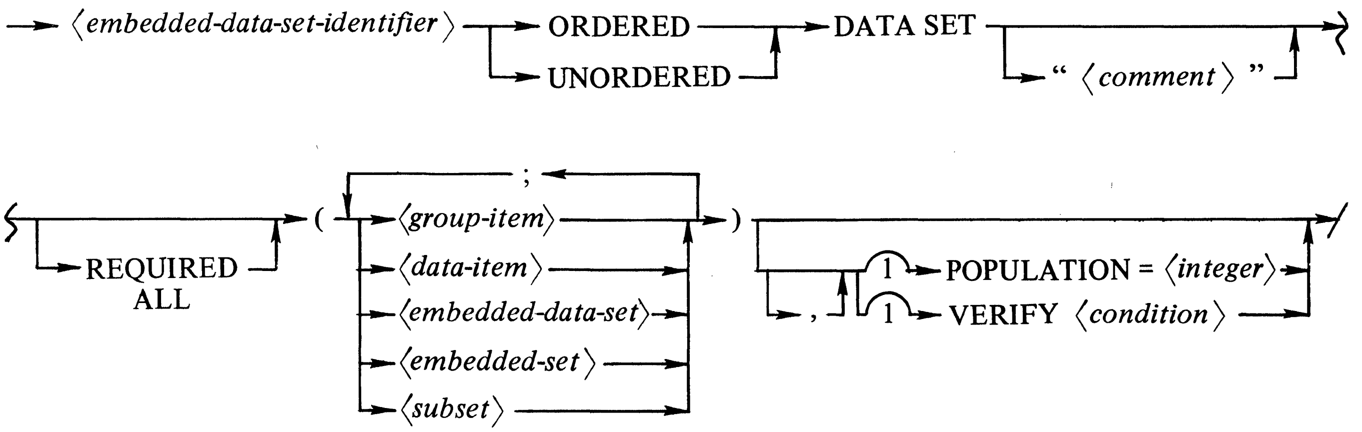
The following example illustrates usage of both a group item and the OCCURS clause.

```
Level 1        GRPA GROUP OCCURS 3 TIMES (
                   DATA1 ALPHA (10) OCCURS 2 TIMES;
  Level 2          DATA2 NUMBER (4) OCCURS 10 TIMES;
                   GRPB GROUP OCCURS 4 TIMES (
                       DATA3 ALPHA (5) OCCURS 3 TIMES;
    Level 3            DATA4 NUMBER (5) OCCURS 3 TIMES;
                       GRPC GROUP OCCURS 5 TIMES (
      Level 4              DATA5 ALPHA (10);
                           DATA6 NUMBER (10))));
```

Embedded Data Set

An item within a data set can itself be a data set; and is referred to as an embedded data set. A data set is used as an item in a data set when it is desired to establish a hierarchical relationship between different types of records. The VERIFY condition and the REQUIRED ALL statements are the mechanism for specifying the minimum criteria that a record must meet prior to its being stored in a data base. These specifications are checked for each record to be stored. The REQUIRED ALL statement specifies that all applicable items must be assigned a non-null value. The VERIFY condition provides the specifications for complex comparisons in determining the validity of a data record.

Syntax



Semantics

a. It is mandatory that the outer level data sets have current records established before lower levels of the structure can be accessed.

b. Either an ORDERED or UNORDERED statement must be specified for an embedded data set.

   1. The ORDERED statement indicates that the data records are to be maintained in sequence. There must be exactly one embedded set declaration for an ordered embedded data set. This declaration specifies the sequence for this data set.

   2. The UNORDERED statement indicates that the system assigns the sequence to the records. No sets are then allowed.

c. When a POPULATION statement is specified for an embedded data set, it indicates the number records per owner. That value is multiplied by the population of the disjoint data set for space allocation. This calculation indicates that the POPULATION specification for the embedded data set should be the average population, rather than the maximum.

d. The REQUIRED ALL statement, if present, is equivalent to specifying the REQUIRED statement for each data item in the data set for which REQUIRED is valid. A value must be present for each of these data items before the record can be stored.

e. The VERIFY condition specifies a certain condition to be satisfied by the items of a potential record to be stored in a data set. If the condition is not satisfied, the record will not be stored.

Example:

```
00002600  :   BOOKS UNORDERED DATA SET(                UNORDERED EMBEDDED
00002700  :         LC NUMBER(9);                          DATA SET
00002800  :         TITLES ALPHA(60);
00002900  :         AUTHR ALPHA (30));
00003000  :   STUDENTS SUBSET OF MSF KEY IS
00003100  :         (LNAME,FNAME) DUPLICATES,
00003200  :         POPULATION = 300)

00007900  :         CORSES ORDERED DATA SET(          ORDERED EMBEDDED
00008000  :               TYPECOURSE NUMBER(1);            DATA SET
00008100  :               YR NUMBER(2);
00008200  :               Q NUMBER(2);                        AND
00008300  :               GCRS SUBSET OF UNIV-COURSES;
00008400  :               GGD ALPHA(2);                  EMBEDDED SET
00008500  :               TITLE-OF-PAPER ALPHA(30);
00008600  :               PPRGD ALPHA(2)),
00008700  :               POPULATION = 4;
00008800  :               CSET ACCESS TO CORSES KEY IS
                              (TYPECOURSE) DUPLICATES)
```

Embedded Set

Embedded sets are used to establish a path of access to embedded, ordered data sets. The embedded set, based on the key value, provides either serial access to all the records of the data set, or random access to a specific record also based on the key value. For embedded, unordered data sets, the only access to the data is based on the physical ordering of the records.

Syntax

——► ⟨*embedded-set-identifier*⟩ ACCESS TO ⟨*embedded-data-set-identifier*⟩ ⟨*key-structure*⟩ ——►/

Semantics

a. The embedded set provides the ability to maintain a logical order for an embedded ordered data set rather than the physical order associated with an embedded unordered data set.

b. One set only, must be declared for each embedded ordered data set. The ACCESS TO declaration is required and must be specified only in this instance. This declaration indicates the establishment of a path, but does not establish physical index tables. (The records are kept in order.)

c. The embedded set and the embedded data set to which it refers must be on the same level.

d. Data items of the key-structure cannot be modified after the record has been stored in the data base. For further information on key-structures, see DISJOINT SET in this Section.

Example:

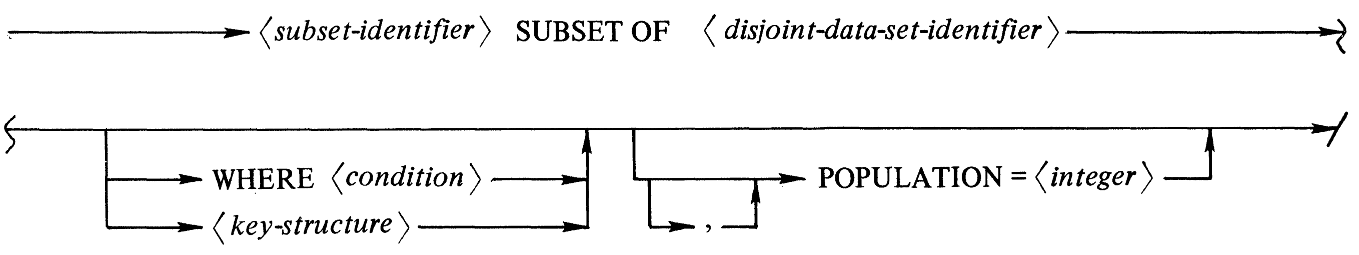
```
CORSES ORDERED DATA SET(
    TYPECOURSE NUMBER(1);
    YR NUMBER(2);
    Q NUMBER(2);
    GCRS SUBSET OF UNIV-COURSES;
    GGD ALPHA(2);
    TITLE-OF-PAPER ALPHA(30);
    PPRGU ALPHA(2)),
    POPULATION = 4;
    CSET ACCESS TO CORSES KEY IS
        (TYPECOURSE) DUPLICATES)
```

Subset

A subset is a path to some of the records of a data set. A subset is not used for holding data values, but rather it provides a method of accessing some of the records of a disjoint data set.

There are two types of subsets, manual and automatic. The manual subset is maintained by the user. It establishes inter-record relationship by providing a method of accessing some of the records of a disjoint data set from records of another data set. The automatic subset is maintained by the system and provides access to records of the disjoint data set which satisfy the condition specified in a WHERE clause.

Syntax



⟨subset-identifier⟩ SUBSET OF ⟨disjoint-data-set-identifier⟩
WHERE ⟨condition⟩
⟨key-structure⟩
POPULATION = ⟨integer⟩
,

Semantics

a. Any subset containing a WHERE clause is an automatic subset.

b. Manual subsets are usually used when the data related to the record is not unique for that record, or when independent access to the related data is required.

c. Manual subsets must be declared as embedded structures. The data set referenced must be a disjoint data set.

d. Manual subsets can have a key-structure. If specified, the key-structure provides serial access based on the key value or random access of a specific record (also based on the key value).

e. Neither the OCCURS clause nor the REQUIRED clause can appear in a subset.

f. Automatic subsets can only be declared as disjoint structures.

g. Automatic subsets must have a key structure, which provides for serial access based on the key value or random access of a specific record (also based on the key value).

h. For further information concerning key structure, see DISJOINT DATA SET in this section.

Example:

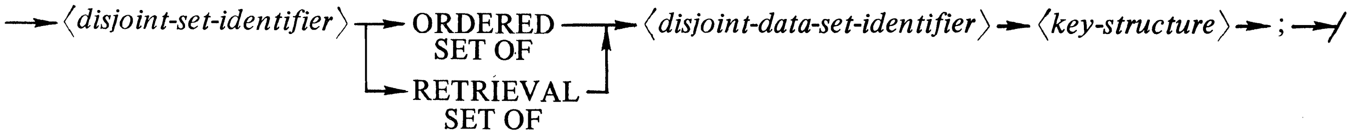
```
00000600  :UNIV-COURSES DATA SET "MAIN FILE"  (
          :
          :
00002500  :        PROFESSOR SUBSET OF UNIV-PERSONNEL, POPULATION  =  3;
          :
          :
00003900  :UNIV-PERSONNEL DATA SET(
          :
          :
00004900  :        COURSES SUBSET OF UNIV-COURSES, POPULATION  =  8;
```

The manual subsets in the above example allow access to the professors who teach a particular course and access to the courses taught by a particular professor. Each professor record could reference all of the courses for it and each course record could reference all the professors for it without any redundancy of data stored.

DISJOINT SET

The use of any set implies the existence of a key. For disjoint data sets, keys can be associated with either an ordered or retrieval disjoint set.

Syntax



Semantics

a. The use of an ORDERED set allows serial access based on the key or random access of a specific record (also based on the key value). The items within the key structure of an ORDERED set specify the control from left to right.

b. If only random access is desired and serial access on the key value if not necessary, a RETRIEVAL set is used rather than an ORDERED set.

c. The DESCENDING clause of the key structure must not be used for items in the key structure of a RETRIEVAL set.

d. If DUPLICATES is not specified in the key structure, the data items of the key structure must not be modified after the record has been stored in the data base.

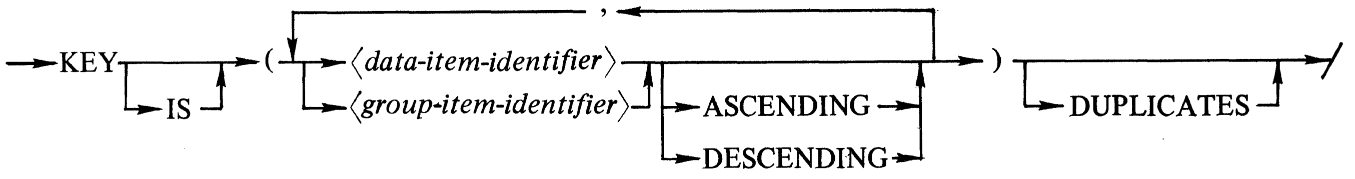
Example:

```
00005200  :     SS-U-P  ORDERED SET OF UNIV-PERSONNEL KEY IS (SSNUM);
00005300  :     U-P-SET  ORDERED SET OF UNIV-PERSONNEL KEY IS
00005350  :          (LASTNAME,FIRSTNAME)  DUPLICATES;
```

Key Structure

The key structure allows the user to identify data items in a data record for which access by a data item value is required.

Syntax



Semantics

a. The key structure consists of a single data item or the concatenation of multiple items.

b. If a group item is used as an item in a key-structure, the COBOL syntax for the random access of a record using that key requires that all the elementary data items of the group be used. The use of a group item in DASDL allows documentation clarity.

c. Each data-item-identifier or group-item-identifier following a KEY specification must refer to a data item or group item of the data set which the set or subset references.

d. To provide flexibility of serial access, each data item may be specified as ASCENDING or DESCENDING. This does not affect random access, and is not allowed on retrieval sets. If neither ASCENDING nor DESCENDING is specified, ASCENDING is assumed by default.

e. Duplicates are records with identical key values. The DUPLICATES option indicates that multiple instances of a key with the same value are allowed. For example, a key NAME normally would allow many John Smiths, but a key of SSUM would not allow any duplicates. A special syntax is provided in COBOL to allow only the retrieval of duplicates.

f. If DUPLICATES is not declared, the key-structure is considered unique.

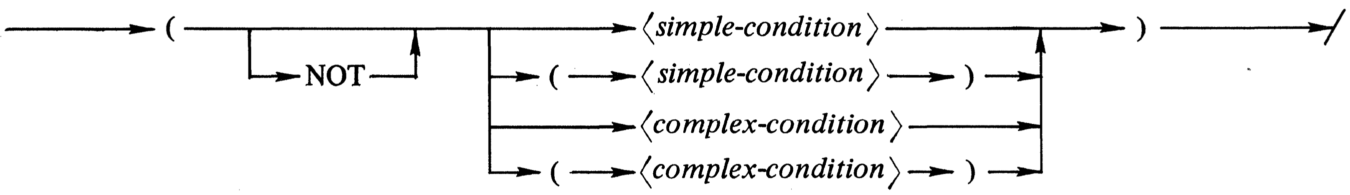
CONDITION

A condition expression has two uses:

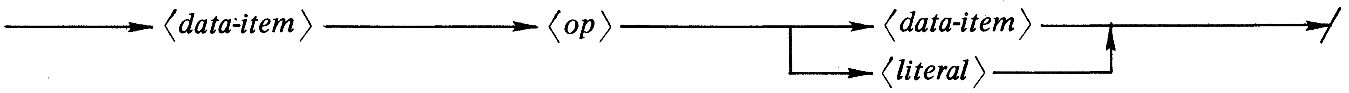
a. Specifying criteria which must be satisfied by a record prior to storage in a data set (VERIFY clause).

b. Specifying the condition for inclusion of a record as a member of an automatic subset (WHERE clause).

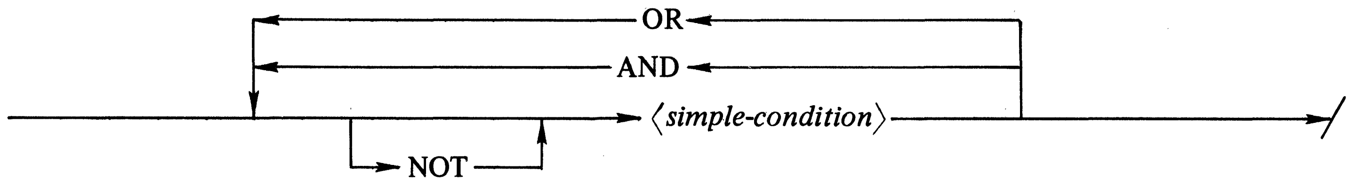
Syntax



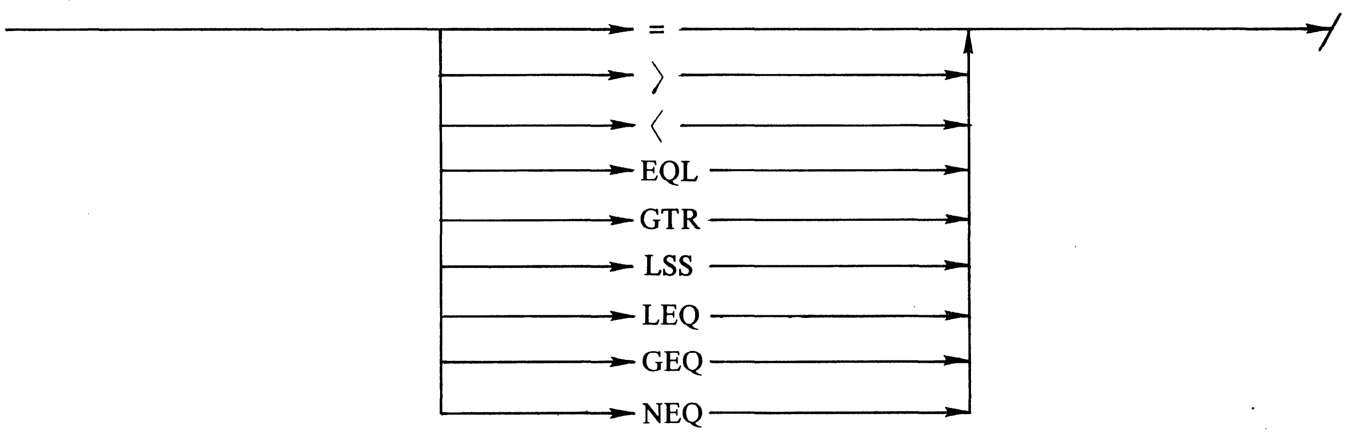
⟨simple-condition⟩



⟨complex-condition⟩



⟨*op*⟩



Condition Semantics

a. Conditions are Boolean-type expressions formed by combining (in a logical and specific manner) data names, literal constants, and relations.

b. The use of parentheses requires a matching left parenthsis and right parenthesis.

c. In any simple-condition having the format

data-name-1 op data-name-2

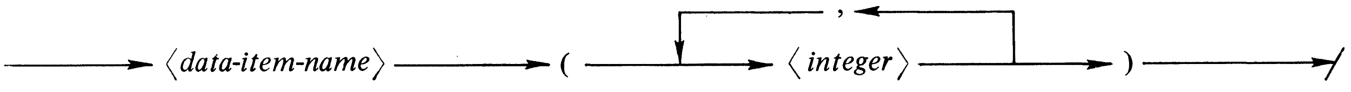
the data items must be of similar type, for example, ALPHA, ALPHA or NUMBER, NUMBER. In comparing alpha-numeric data items, the comparison is based on the longest field. The shorter field is compared as if it were blank-filled to the right. In the example below, A will be equal to B if the most significant four characters of A and B are the same and the last two characters of B are blanks. For example:

```
A ALPHA (4);        % "ABCD"
B ALPHA (6);        % "ABCDƀƀ"
```

decimal aligned ?

NUMERIC-defined items are compared on the numeric value, independent of the length of the items.

d. When data items are defined within the scope of an OCCURS clause, all necessary subscripts must be specified.



Example:

simple-condition
STUAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 1)

complex-condition
WHERE (SALARY LSS OR SALARY EQL 0)
SEXSET SUBSET OF MSF WHERE (SAGE 21 AND NOT SSEX)
VERIFY ((HOURSCRDT GTR 0 AND CLASS-SIZE LEQ 60) AND NOPROF NEQ 0) ;

PHYSICAL STRUCTURES

The data base structures can be directed to system disk or removable user disk. The data base structure files differ from the standard B 1700 system files as follows:

a. DMSII is responsible for the allocation and maintenance of data space.

b. DMSII control information can be appended to the data in the records.

c. Multiple data base structures can be mapped into one file.

There are four types of DMSII physical structures:

a. Data Set.

b. Index Sequential.

c. Index Random.

d. List.

    1. Ordered.

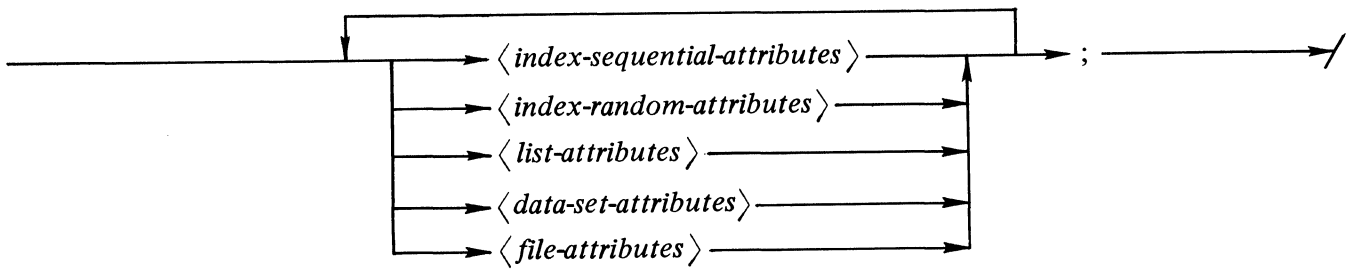
    2. Unordered.

Logical structures are mapped into one of the above four types of physical structures, and are mapped according to the list below.

Logical Structure	**Physical Structure**
Disjoint Data Set	Data Set
Ordered Set	Index Sequential
Retrieval Set	Index Random (Index Sequential Optional)
Embedded Data Set	
Unordered	Unordered List
Ordered	Ordered List
Manual Subset	
No Key	Unordered List
With Key	Ordered List
Automatic Subset	Index Sequential

Unless overridden by explicit file attributes, file names are created by using the default-naming conventions described in appendix A.

Syntax



Semantics

a. Physical descriptions allow distinction between logical and physical structures in the data base description.

b. Physical descriptions must be declared at the outer level only.

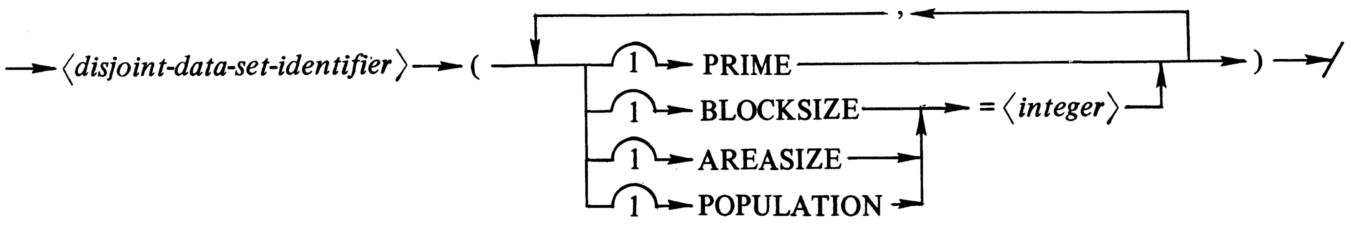
c. Physical descriptions must refer to sets, subsets, or data sets that have been defined previously.

d. By not using a physical description, default values are assigned for the physical properties of the data base if no explicit specifications are made. If explicit assignment is made, the DASDL compiler will not change the assigned value.

Disjoint Data Set Attributes

Each disjoint data set is allowed to have a default physical structure built by not listing any physical structure attributes. Disjoint data sets can have one or more sets associated with them, but only one structure can be responsible for the allocation and de-allocation of physical space for the data records. The structure controlling space is called the PRIME structure; it is either the data set itself or any of the sets of the data set. When a set is used as PRIME and that set with its data set are stored on a moveable head storage device is one significant use of the PRIME structure. By making a set the PRIME structure, it is generally true that with one movement of the read/write head, the system can obtain both the table and the data when one file is used to hold both structures. Each file area is partitioned into a section for the index tables, and a section for those data records whose index entries are in the index tables of the same area. Under all circumstances, PRIME set accessing has been optimized to provide performance benefits over non-PRIME accessing.

Syntax



Semantics

a. There is only one PRIME structure per data set and its associated structures.

b. The BLOCKSIZE, AREASIZE, and POPULATION statements are specified in number-of-records.

c. AREASIZE must be greater than 1 and greater than or equal to BLOCKSIZE.

DEFAULT VALUES

POPULATION	10000
BLOCKSIZE	If the record size is less than or equal to 720 bits, the size is 1440 bits divided by the record size. If not less than or equal to 720 bits, BLOCKSIZE is equal to 1.
AREASIZE	Maximum (POPULATION/20) or 50.
PRIME	FALSE, if there are no sets.

The system makes the first ordered set of a data set by default PRIME. If there are no ordered sets, the system selects the first retrieval set. If there are no sets, the data set itself is considered PRIME.

Data Set Maintenance Techniques

If the data set is specified or defaulted as PRIME, space is maintained by a Next Available counter and a Highest Open counter for all areas of the physical file. If the Next Available counter equals the Highest Open counter, the record is stored at the record address indicated by Next Available and Highest Open counter. Both counters are then incremented by 1. If the Next Available counter is unequal to the Highest Open counter, the address in which to store the data record is taken from the Next Available counter, which is then set to the contents of the record at the address. (Available records are linked together.)

If the data set is not specified or defaulted as PRIME, there is a Next Available counter plus a Highest Open counter for each area of the physical file; the area of the index file in which the key is inserted for the PRIME set, determines which pair of counters to use.

Example:

PRIME Data Set

AREA 0	AREA 1
REC 0	REC 5
REC 1	REC 6
REC 2	
REC 3	
REC 4	

Next Available counter = 7
Highest Open counter = 7
If REC 2 is deleted, Next Available counter = 2
and Highest Open counter remains 7.

Non-PRIME Data Set

AREA 0	AREA 1
REC 0	REC 5
REC 1	REC 6
REC 2	
REC 3	

For AREA 0:	Next Available counter = 4 Highest Open counter = 4
For AREA 1:	Next Available counter = 7 Highest Open counter = 7

If REC 2 is deleted, then Next Available counter = 2 for AREA 0. Next Available counter and Highest Open counter for AREA 1 remain the same.

Index Sequential Attributes

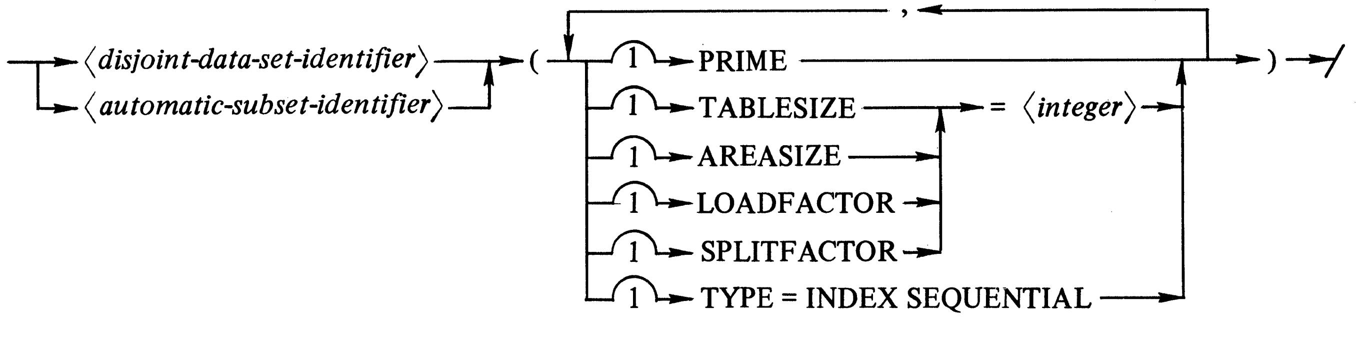
INDEX SEQUENTIAL is the structure used to map disjoint sets and automatic subsets. The algorithm used maintains multiple levels of tables called coarse and fine tables. A coarse table is split when it becomes full, resulting in multiple levels of coarse tables. DASDL provides the user with control of the tables using the following two parameters: LOADFACTOR and SPLITFACTOR.

LOADFACTOR entries can be placed in an existing fine table before a new fine table is started. This applies only when the new entries are added at the end of the existing entries.

It is advisable to leave additional space in tables so when records are inserted after the initial load, there is space for the new entries without having to create new tables. The SPLITFACTOR determines how many coarse table entries are moved when it becomes necessary to split a coarse table. The number of entries specified by SPLITFACTOR are moved to a new coarse table; this new table is a new level of coarse table, with a new entry referencing it in the split coarse table.

The loading of this structure should be done in the sequence described by the key structure in order to optimize the access of entries in the table.

Syntax



Example:

```
00003700  :         POPULATION = 1000;
00003800  :         UNIV-C-SET ORDERED SET OF UNIV-COURSES KEY IS (CRS-NO);
          :
          :
          :
00011000  : UNIV-C-SET(
00011100  :         TABLESIZE = 12,
00011150  :         AREASIZE = 10,
00011200  :         TYPE = INDEX SEQUENTIAL,
00011300  :         LOADFACTOR = 9);
```

Semantics

a. The AREASIZE specifies the number of tables per area.

b. The TABLESIZE specifies the number of entries per fine table. Coarse table size is set to the number of coarse table entries that can fit into the same amount of space.

c. The LOADFACTOR specification is a percentage of entries per fine table and must be greater than zero.

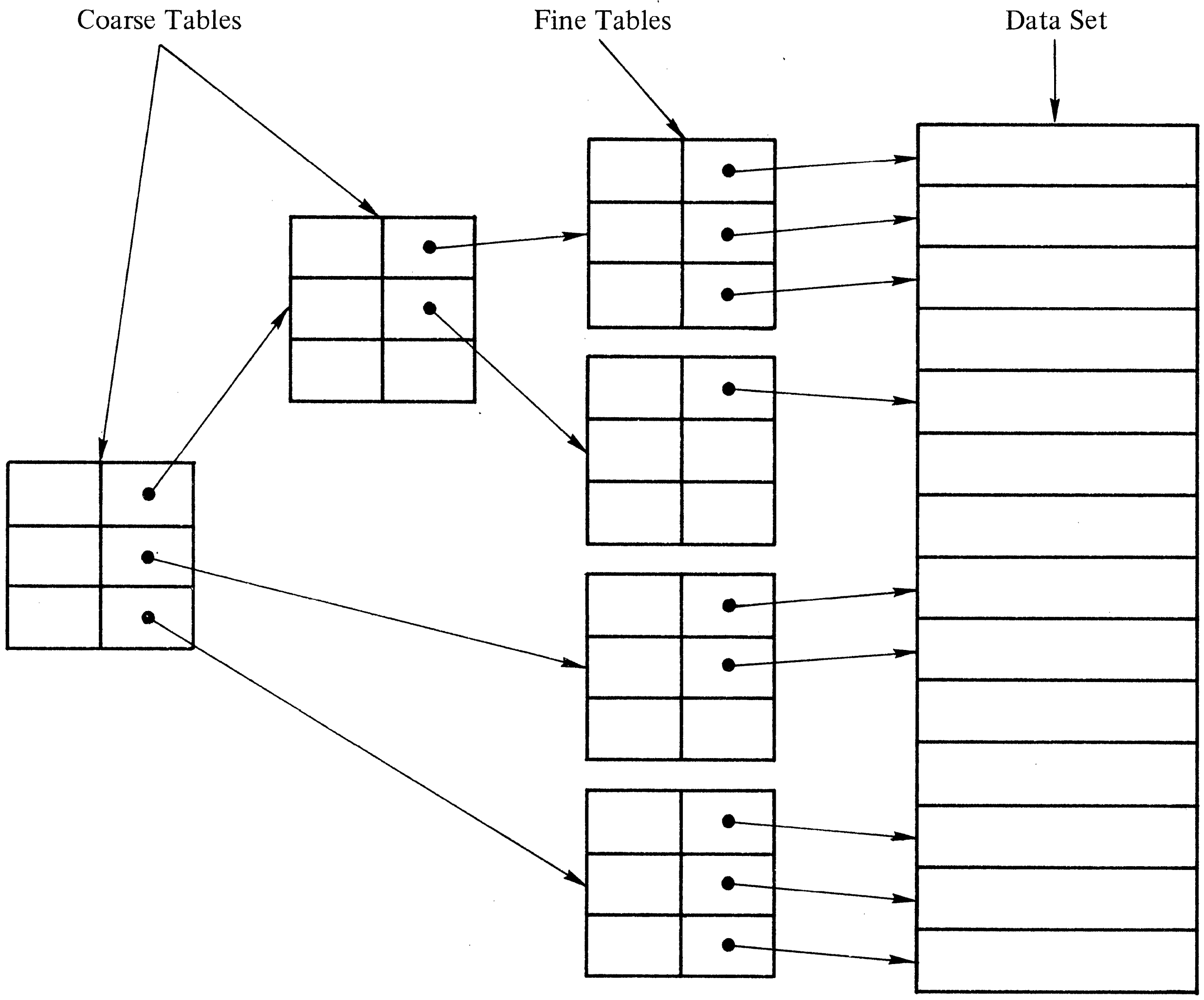
The SPLITFACTOR specification is a percentage of entries per coarse table and must be greater than one.

d. The TYPE specification is optional unless the disjoint set-identifier refers to a retrieval set, then it is required.

DEFAULT VALUES

PRIME	TRUE (if this is the first, or only, ordered set).
AREASIZE	(AREASIZE of data set) + 2.
TABLESIZE	The square root of POPULATION.
LOADFACTOR	66% of TABLESIZE (fine).
SPLITFACTOR	50% of TABLESIZE (coarse).

Example:



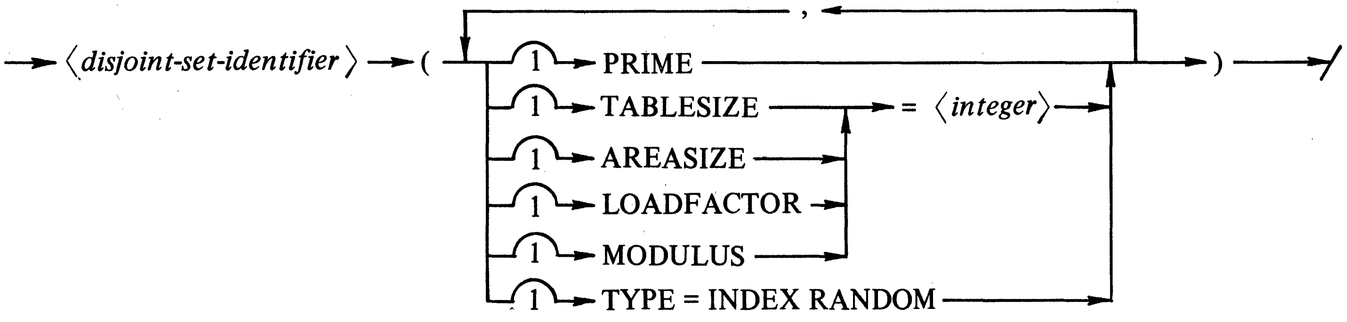
NOTE

Coarse tables can only go the three levels; fine tables are limited to one level. Coarse tables always contain the highest valued key of the next lower table.

Index Random Attributes

An index random structure is built by default for each disjoint retrieval set in a data base. The algorithm takes the symbolic key in a hashed format, performs a remainder divide by the number of base tables (MODULUS), then searches the resulting table. When any of the base tables becomes full, additional entries for that table are placed in overflow tables. Enough space in the base tables should be allocated to minimize table overflow.

Syntax



Example:

```
:    STUSET RETRIEVAL SET OF STUDENT KEY (IDNO);
:
:
:
:    STUSET (MODULUS = 3, LOADFACTOR = 1);
```

Semantics

a. The MODULUS statement specifies the number of base tables. The MODULUS specification must be less than or equal to the AREASIZE specification multiplied by the number of areas but greater than 1.

b. The LOADFACTOR specification is a percentage that controls the distribution of overflow entries into overflow tables. If the LOADFACTOR entry implies one entry, only one base table overflows into any given overflow table. The LOADFACTOR entry must be greater than zero, and less than or equal to TABLESIZE.

c. The TABLESIZE specification is the number of entries each table can hold.

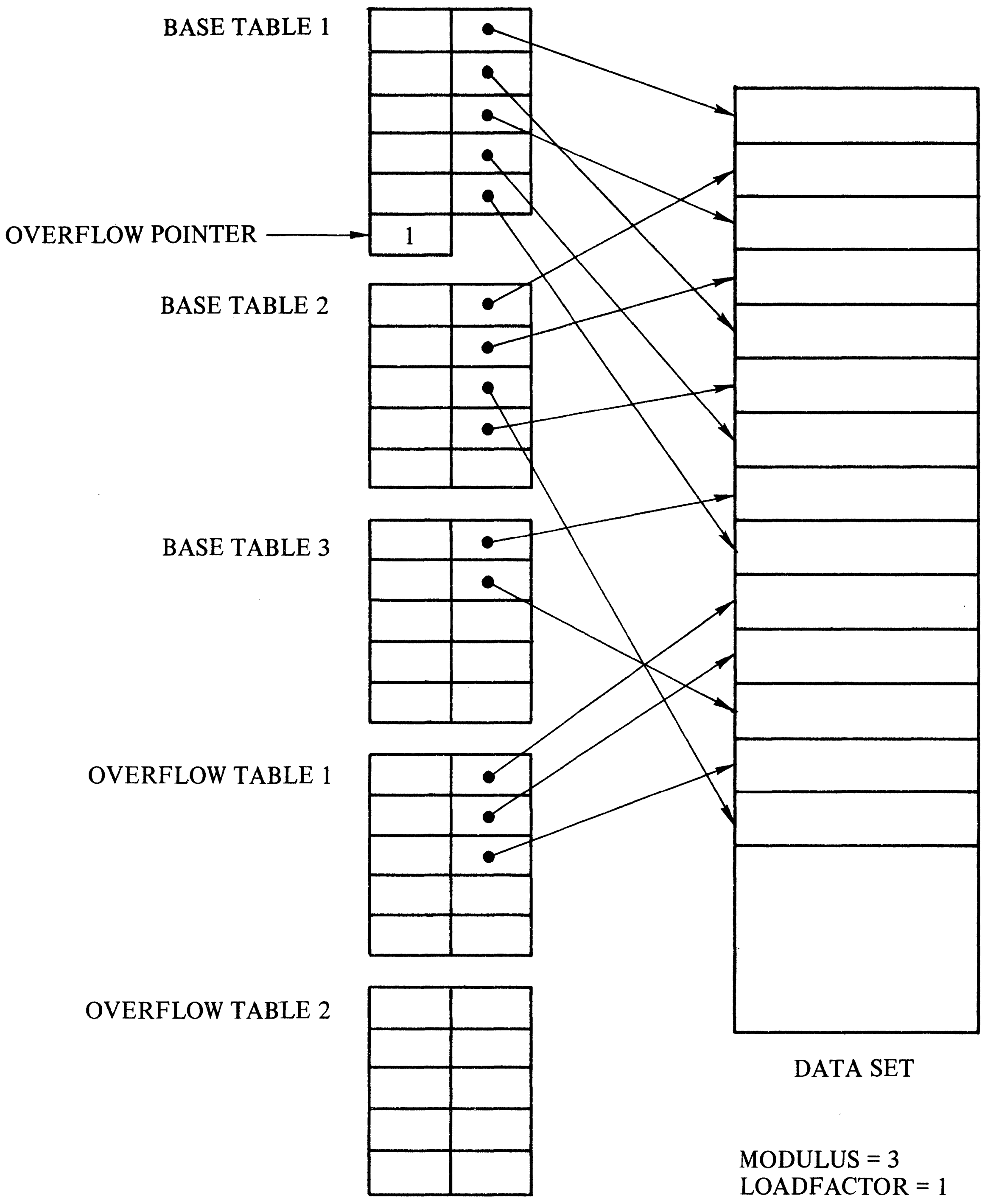
d. The AREASIZE specification is the number of tables per area.

e. The TYPE specification is optional.

DEFAULT VALUES

PRIME	True (if this is the first, or only, retrieval set and there are no ordered sets).
MODULUS	The square root of POPULATION multiplied by 1.1.
TABLESIZE	MODULUS.
LOADFACTOR	66% of the MODULUS value.
AREASIZE	Maximum(MODULUS/15) or 10 if PRIME is true; otherwise, Maximum (MODULUS/5) or 10.

Example:

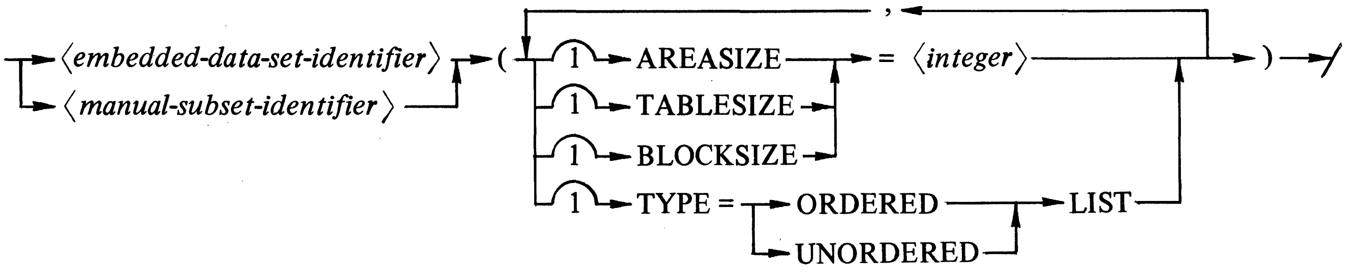

BASE TABLE 1
OVERFLOW POINTER
1
BASE TABLE 2
BASE TABLE 3
OVERFLOW TABLE 1
OVERFLOW TABLE 2
DATA SET
MODULUS = 3
LOADFACTOR = 1

List Attributes

An ORDERED LIST is built for each ordered embedded data set, and for each manual subset with a key. An UNORDERED LIST is built for each unordered embedded data set and for each manual subset without a key. Each record in a data set containing an embedded data set or a manual subset requires greater storage space than the data record requires, since control information is appended. The storage required for records of embedded data sets is also increased due to control information. Additional storage for the embedded data set can be reduced by placing multiple data records in a table, since control information is only stored once per table.

An ORDERED LIST is maintained by placing records in physical order. Serial access becomes more efficient as the number of entries per table increases.

Syntax



Example:

```
00002600  :         BOOKS UNORDERED DATA SET(
          :
          :
          :
00010500  : BOOKS(
00010600  :         AREASIZE = 500,
00010650  :         TYPE = UNORDERED LIST,
00010700  :         BLOCKSIZE = 5);
```

Semantics

a. The TABLESIZE attribute is the number of records per table.

b. The BLOCKSIZE attribute is the number of tables per block.

c. The AREASIZE attribute is the number of tables per area. AREASIZE must be greater than or equal to the BLOCKSIZE entry.

d. Each table may contain records from only one owner.

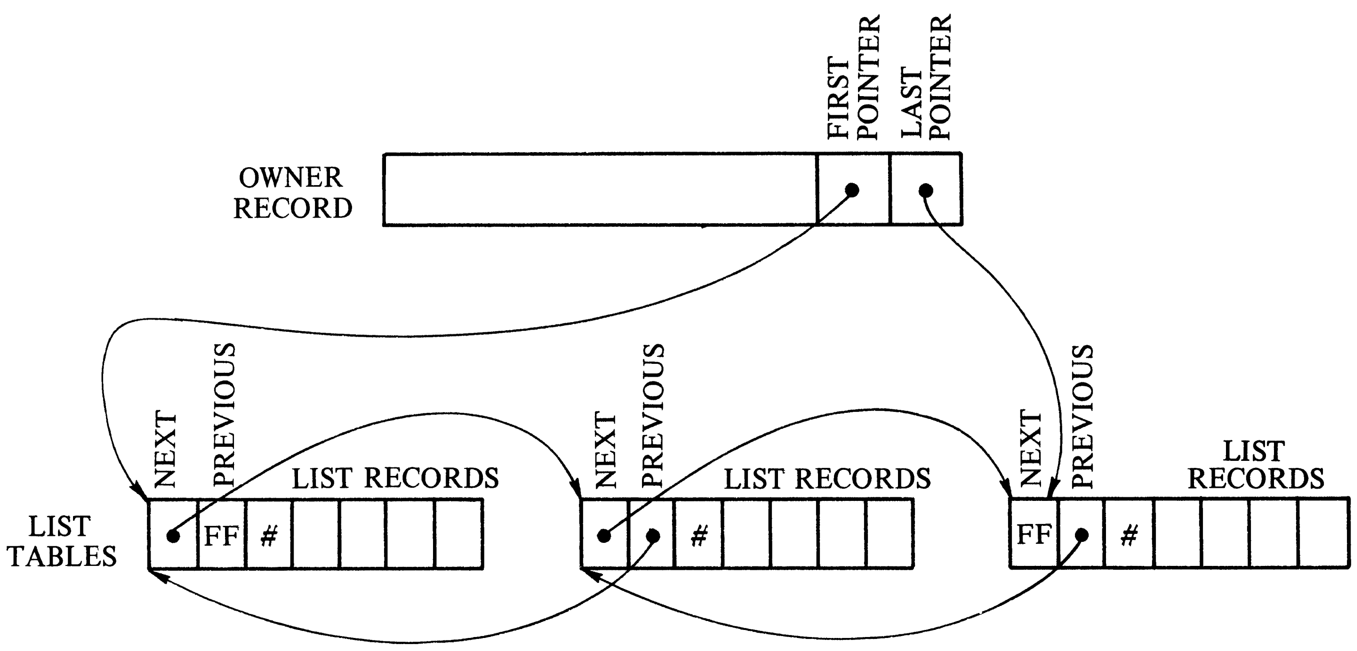
e. The TYPE entry is optional.

DEFAULT VALUES

TABLESIZE	(Maximum(number of records plus the control information size that fits into one segment, or 1.)
BLOCKSIZE	Maximum (number of tables that fit into one disk segment or 1)
AREASIZE	((POPULATION * owner's POPULATION) /TABLESIZE) / AREAS

Example:

LIST STRUCTURE



FF = Null value.

# = Number of LIST records containing information.

TABLESIZE = 4.

File Attributes

Storage files can be specified at DASDL compilation time either to alter the title of a file, assign pack storage, alter areas, or group logical structures into a single physical file.

The structures that can be stored in a single physical file are listed below. However, no structure can be in more than one structure list. Any one of the following items a. through f. can be stored in a single physical file.

a. A disjoint data set.

b. An embedded data set.

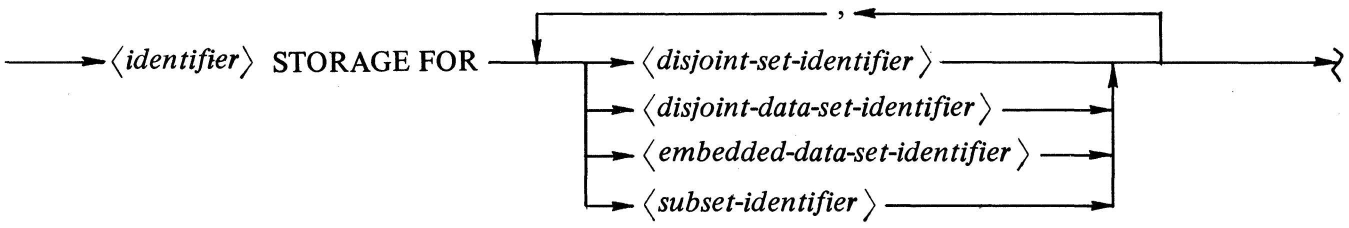
c. A disjoint data set plus one of its sets.

d. Up to 16 index sequential sets or automatic subsets.

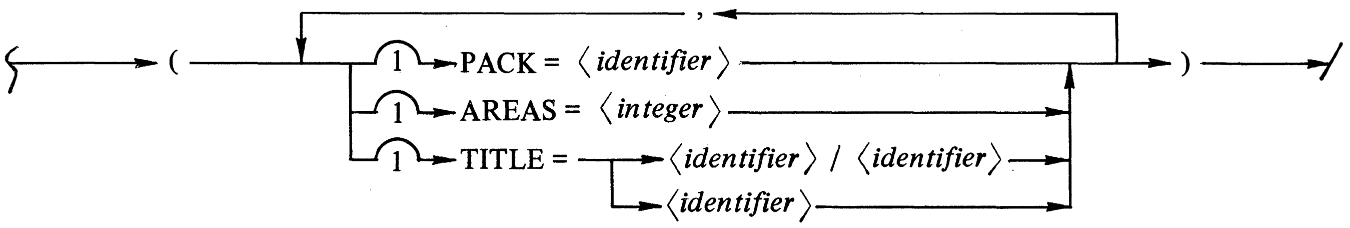
e. One index random set.

f. A manual subset.

Syntax



⟨identifier⟩ STORAGE FOR
,
⟨disjoint-set-identifier⟩
⟨disjoint-data-set-identifier⟩
⟨embedded-data-set-identifier⟩
⟨subset-identifier⟩



Example:

```
00010800    :BOOKFILE STORAGE FOR BOOKS(
00010850    :       TITLE - UNIV/LIBRARY,
00010900    :       AREAS = 10);
```

Semantics

a. If a PACK entry is specified, it must be a valid user pack identifier.

b. The number of areas assigned by the AREAS specification must be greater than zero, and less than or equal to 105. AREAS is the maximum number of areas in the disk file.

DEFAULT VALUES

TITLE	The STORAGE identifier.
PACK	NULL.
AREAS	20.

4. COBOL INTERFACE

INTRODUCTION

There are two interfaces between the host language, COBOL, and the data base system: one during compilation and one during execution. The compilation interface provides syntax allowing an application program, through the use of the INVOKE statement, to use any or all portions of a data base. The invoke process consists of utilizing DASDL-generated library files that supply the COBOL compiler a description of the user-selected portions of the data base. The COBOL compiler then compiles an appropriate execution-time interface with the data base.

The execution interface consists of a number of record areas, one for each data set invoked, and a number of paths, one for each set or subset.

Associated with every record area is a current-record pointer. A record is considered to be the current record of a data set if the appropriate current-record pointer refers to an existing record in the data base. Usually, the record area contains a copy of that record, at least until the record area is changed by the program.

The current-record pointer for a data set is changed by any operation that causes a new record to be placed in the record area, or placed into the data base from the record area. The establishment of any record as the current record for a record area, through use of a MODIFY or STORE operation, locks the record, making that record unavailable to any other user. Changing the current record pointer automatically unlocks any previously locked record and if required locks the new one.

Sets and subsets are represented as paths rather than as records. Their purpose is to locate the records of the data set with which they are associated. The current-path pointer associated with every set and subset (but not data set) refers to the last record accessed by way of that set or subset. This current-path pointer retains its reference until explicitly changed, or until the record referenced by the current-path pointer is deleted from the data base.

A current-record pointer can be in one of four states:

a. Undefined state – not valid for any purpose. For example, just after a data base OPEN.

b. Created state – indicates a CREATE operation has just been executed for a data set. If a STORE is the next operation to be executed against the data set, a new record is stored. In all other instances the created state is the same as the undefined state.

c. Defined state – refers to a valid record. For example, current record pointer is defined after a successful FIND operation. A record can be locked only if it is in this state.

d. Deleted state – indicates there is no valid current record, but the current-record pointer maintains a position in the data set. For example, current-record pointers are in the deleted state following a DELETE operation since the current record has been removed. A current-record pointer in the deleted state can be used to access the next or prior record but not a current record.

Figure 4-1 illustrates the current-record pointer states.

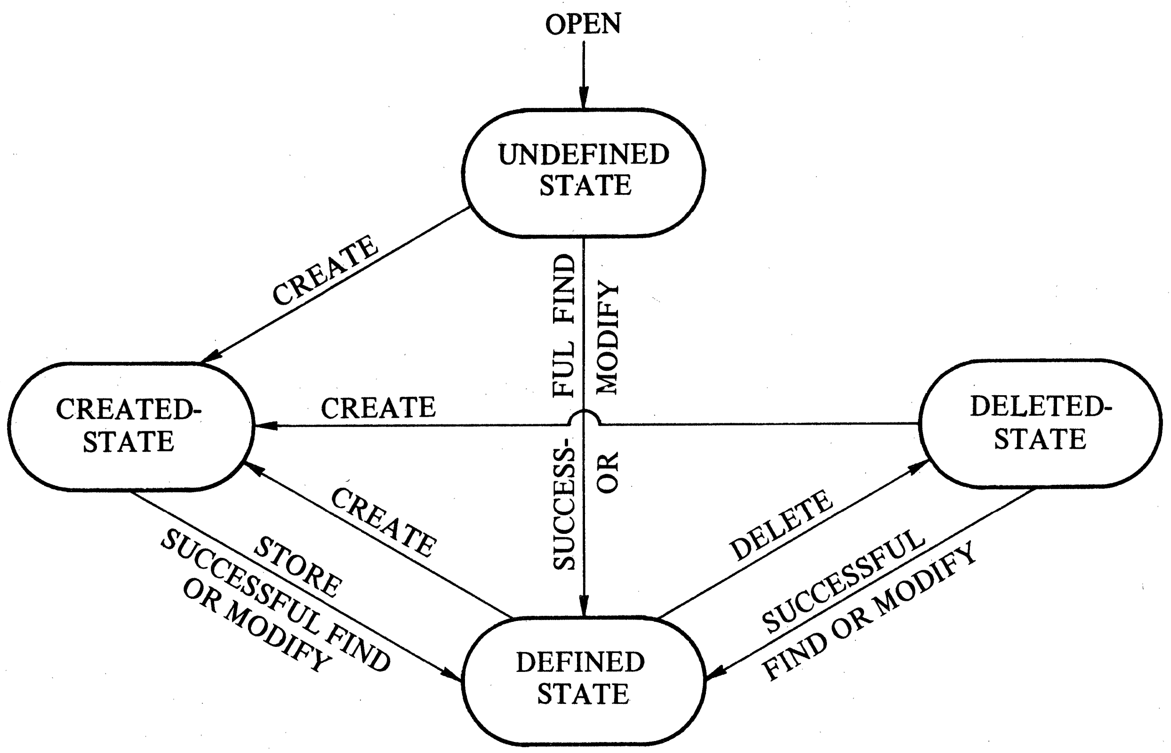


Figure 4-1. Current-Record Pointer State

The functional operation of figure 4-1 is explained below:

a. When the data base is opened, the current record pointer is in the undefined state.

b. An unsuccessful FIND or MODIFY operation does not change the current-record-pointer state.

c. A RECREATE operation affects the current-record-pointer state the same as the CREATE operation.

d. Whenever the current-record pointer of a data set changes, the current-record pointer for all embedded data sets becomes undefined.

e. A FREE operation does not affect the state of the current record pointer.

f. A current record pointer can be set to the deleted state due to the actions of some other program being multiprogrammed against the same data base.

A set or subset current path pointer can be in one of three states:

a. Undefined.

b. Defined.

c. Deleted.

Figure 4-2 illustrates a set or subset current-path pointer processing state.

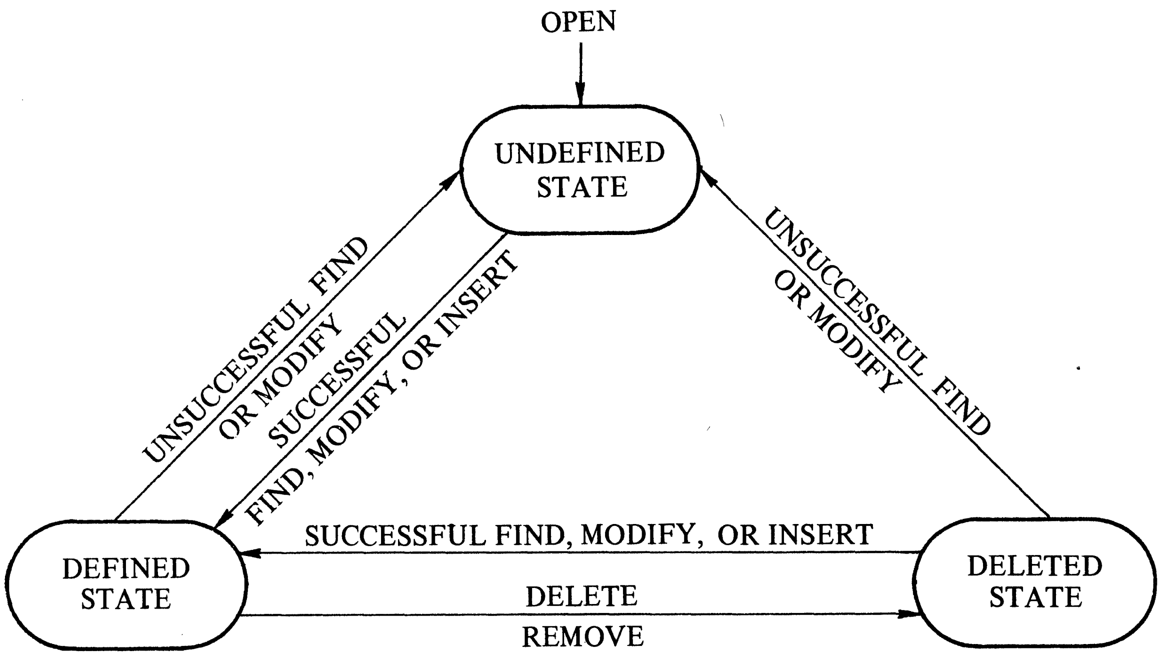

OPEN
UNDEFINED STATE
UNSUCCESSFUL FIND OR MODIFY
SUCCESSFUL FIND, MODIFY, OR INSERT
UNSUCCESSFUL FIND OR MODIFY
DEFINED STATE
SUCCESSFUL FIND, MODIFY, OR INSERT
DELETE
REMOVE
DELETED STATE

Figure 4-2. Set Or Subset Current-Path Pointer State

The functional operation of figure 4-2 is explained below:

a. When a data base is opened, the set or subset current path pointers are in an undefined state.

b. An unsuccessful FIND or MODIFY operation, by way of a current-path pointer of a set or subset, changes the set or subset current-path pointer.

c. The STORE, CREATE, RECREATE, and FREE operations do not affect a set or subset current-path pointer.

d. An INSERT operation always changes the current-path pointer of a subset to the defined state.

e. Whenever the current-record pointer of a data set changes, the current-path pointer for all of that data set's embedded sets and subsets become undefined.

f. A set or subset current-path pointer can be set to the deleted state as the result of another program accessing the same data base.

COBOL DATA DIVISION

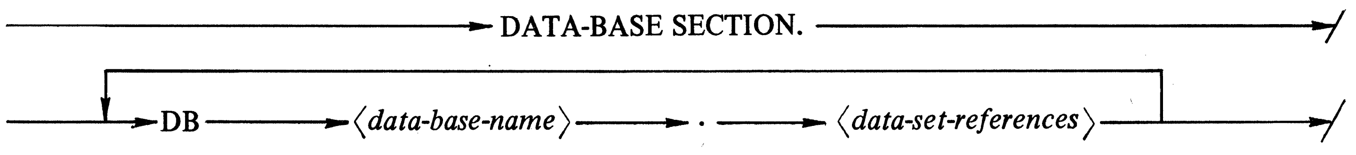
General

A DATA-BASE SECTION must be inserted within the DATA DIVISION of a COBOL program supplying the COBOL compiler with a description of all or selected portions of a data base. The DATA-BASE SECTION is placed between the FILE SECTION and the WORKING-STORAGE SECTION.

DATA-BASE SECTION

In the DATA-BASE SECTION all data sets intended for use are invoked. This signals the compiler to include in the compilation the item names and all path names (sets and subsets), plus all embedded data sets and subsets within the invoked data set. The compiler also establishes the necessary user record areas.

Syntax



Example:

```
001031    DATA-BASE SECTION.
001032    DB    UNIV.
```

Semantics

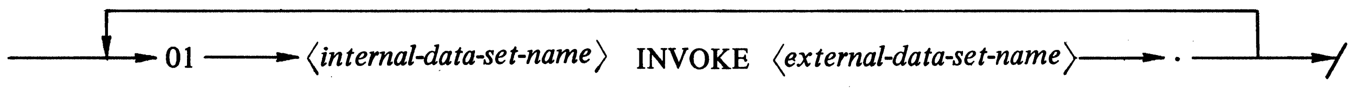
a. The level indicator, DB, is used to select a particular data base. Any particular data base can be referenced only once per program, and only one data base can be open at any one time.

b. The data-base-name identifier can be used as a qualifier of data sets or set names. The data-base-name is the family-name of the program-identifier used in the DASDL compilation (see appendix A).

Data Set References

The referenced data base can be followed by any number of data set references.

Syntax



Example:

```
001033    01 MASTER INVOKE MSF.
001034    01 ADDRESS INVOKE ADR.
```

Semantics

a. The level number 01 is used to select particular data sets from a data base.

b. Each compilation copies the description of each invoked data set into the program from a library file created by DASDL. The file-identifier of this library file for each data set has the following format:

   data-base-name / data-set-name

c. The internal-data-set-name allows synonym capability and can also be used to establish more than one record area for a data set.

d. All references to the data set in the program are by the internal-data-set-name. The internal-data-set-name can be a name assigned by the programmer, or the name of a data set defined in DASDL. The internal-data-set-name must be different than the external-data-set-name if the data set is invoked more than once within a program. The use of the internal-data-set-name provides a unique name for each record area of the data set, and is required only if the data set is invoked more than once.

e. Embedded data sets must not be programmatically invoked. They are automatically invoked when the data set to which they belong is invoked.

f. All disjoint data sets, if used, must be invoked. This method also applies to any disjoint data sets referenced by a subset if the subset is used.

INVOKED DATA SET

The COBOL compiler prints the names of all the paths and data items, and also shows the structure number (DDL-NUMBER) assigned at the DASDL compilation. The source statements supplied by the DASDL compiler are distinguished from the COBOL source statements by an asterisk (*) appearing to the left of the print line, as the coding example below indicates.

Example:

```
 001034    01 ADDRESS INVOKE ADR.
*
*          01 ADR DATASET DDL-NUMBER 10 20: 0: 4 12/ 5/74
*            ORDERING KEY SSAD DDL-NUMBER 19 20: 0: 4 12/ 5/74
*                 (SNO).
*              02 FACULTY-STUDENT                      PIC 9 COMP.
*              02 SNO                                  PIC 9(9) COMP.
*              02 ADLN OCCURS 9 TIMES                  PIC X(54).
*              02 ZIPC                                 PIC 9(5) COMP.
*              02 PHON                                 PIC 9(10) COMP.
*
```

The structure number, along with an internally assigned invoke number, allows the system to update the correct record areas. Even when the structure number is the same, the invoke number ensures that the correct record area is altered. The level numbers generated by the COBOL compiler reflect the usage of data items by level indicators. The listing also displays the time and date the files were created by the DASDL compilation.

MULTIPLY-INVOKED DATA SET

Since one record area can only hold one record at a time, it may be necessary, for effectiveness, to have more than one record area. In the following example, MSF is invoked twice, creating two separate record areas for MSF so that two different records of MSF can be used at the same time. This example provides multiple current records.

The following example also provides multiple current path pointers for the same set. Each current path pointer is updated only when explicitly used. Either record area can be updated by any of the paths to MASTER or FILE1.

```
DATA-BASE SECTION
DB UNIV.
01  MASTER INVOKE MSF.
01  FILE1 INVOKE MSF.
```

COBOL PROCEDURE DIVISION

General

The DATA-BASE SECTION allows the invocation of all or part of a data base. As a result of the invoked description, the compiler generates the necessary interfaces, so that at data base open time, the proper data set record areas are allocated.

Special extensions to COBOL are used to manipulate data sets. Data base retrieval and storage are accomplished at the record level, with one record being transferred into or out of the record area together with selected data base operations.

Move And Move Corresponding

The record area for a data set contains two types of items: one type is control information, the other is the data. The portion containing data items is similar to a WORKING-STORAGE 01 entry indicating that all COBOL data manipulation statements can be utilized in the moving of data items. This includes the group MOVE and the MOVE CORRESPONDING operations as the following example illustrates.

```
*        01  MSF DATASET DDL-NUMBER            6  20:  0:  4  12/  5/74
*         ORDERING KEY MSFSET DDL-NUMBER           18  20:  0:  4  12/  5/74
*              (SSNO).
*          02  SSNO                                     PIC  9(9)  COMP.
*          02  NONAM                                    PIC  9  COMP.
*          02  LNAME                                    PIC  X(30).
*          02  QUARTER DATASET DDL-NUMBER        15  20:  0:  4  12/  5/74
*            ORDERING KEY QSET DDL-NUMBER        15  20:  0:  4  12/  5/74
*                (QTR).
*            03  QTR                                    PIC  X(4).
*            03  QTTRHRS                                PIC  99  COMP.
*            03 QTRQP                                   PIC  99  COMP.
```

The functional description of the above example is explained in the following list.

a. MSFSET, QUARTER, and QSET are control items and are not moved in a MOVE MSF TO . . . or a MOVE . . . TO MSF operation.

b. QTR, QTTRHRS and QTROP are items of the record area for QUARTER and therefore are not moved in a MOVE MSF TO . . . or a MOVE . . . TO MSF operation.

c. The MSF record area for a group MOVE operation can be considered as the following items:

```
01    MSF
   02 SSNO
   02 NONAM
   02 LNAME
```

d. Items SSNO, NONAM, and LNAME are the only candidates for a MOVE CORRESPONDING operation.

e. A group MOVE operation is always considered as an alphanumeric MOVE.

Exception Processing

The COBOL PROCEDURE DIVISION has been extended by adding DMSII statements, providing an interface between a COBOL program and a data base. The system, when executing DMSII statements, can encounter any one of several exception conditions that prevents the operation being performed as specified.

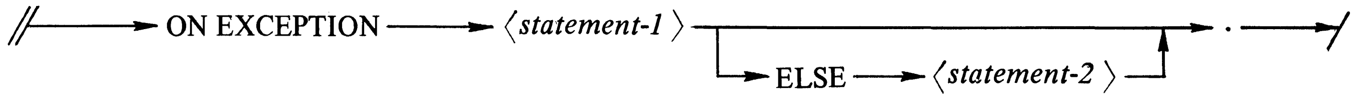
If an exception condition occurs, the program terminates unless the DMSII statement is followed by an ON EXCEPTION clause. It is recommended, therefore, that the ON EXCEPTION clause be used following DMSII statements.

To further qualify the nature of an exception, there exists for each COBOL program a special register: DMSTATUS. DMSTATUS is set by the system at the completion of each DMSII statement.

ON EXCEPTION CLAUSE

The syntax chart notation indicates that an ON EXCEPTION clause may appear, by the presence of a double slash (/ /). The ON EXCEPTION clause syntax that follows a double slash is shown below.

Syntax



The following example illustrates the ON EXCEPTION programming technique:

Example:

```
STORE CORSES ON EXCEPTION PERFORM STATUS-BOOLEAN.

MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION
IF DMSTATUS(NOTFOUND) DISPLAY C-SSNO "NOT IN MSF" ELSE
PERFORM STATUS-BOOLEAN.
```

Semantics

a. Each DMSII statement yields a true/false value which is true if the operation resulted in an exception condition; the value is false if the operation completed with no exceptions encountered. If true, statement-1 of the ON EXCEPTION clauses will be executed; otherwise, statement-2 will be executed if present.

b. Logically, DMSTATUS can be used to qualify an ON EXCEPTION clause.

c. If the ON EXCEPTION clause is not specified, the occurrence of an exception terminates the program.

DMSTATUS Register

The DMSTATUS register provides the capability to determine the nature of an exception should an exception occur. DMSTATUS is set by the system at the completion of each DMSII statement, and is used to qualify an ON EXCEPTION clause. To isolate the exception encountered, a number of attributes exist for DMSTATUS. Each attribute yields a Boolean value to indicate whether that particular category of exception has occurred. The DMSTATUS register, when used, has the following format.

DMSTATUS Syntax

```
──────────────►DMSTATUS─────► ( ⟨category-name⟩ ) ──────────────────►/
```

The category-name and its descriptions are listed in table 4-1.

Table 4-1. DMSTATUS Categories and Description

Category Name	Exception Condition Description
NOTFOUND	This record does not satisfy a SELECTION expression such as in MODIFY or FIND.
	Key value in record does not match key of a manual subset.
	No current record exists (previously deleted).
	Current-record pointer is undefined.
	Either the master record is undefined or the embedded structure is empty.
DUPLICATES	Duplicates not allowed in set (STORE operation).
	Duplicates not allowed in a manual subset (INSERT operation).
DEADLOCK	A "deadly embrace" condition has occurred while trying to lock records. Note that the system has automatically performed a FREE operation of all records for this program.
DATAERROR	An attempt was made to store a record with a null key or null required item.
	An attempt was made to store a null record (all bits = 1).
	A DASDL verify condition not met.
NOTLOCKED	A STORE statement not preceded by a CREATE, RECREATE, MODIFY, or STORE.

Table 4-1. DMSTATUS Categories and Description (Cont)

Category Name	Exception Condition Description
KEYCHANGED	An attempt was made to store a record when the value of an item, used as a key in a set, was illegally changed (duplicates not allowed, or embedded set).
SYSTEMERROR	Only one data base can be open at a time.
IOERROR	An I/O error was encountered trying to read from or write to the data base.
LIMITERROR	Data exceeds the size of physical structure.
OPENERROR	Data base not initialized.
	Already open.
	Run-time description of data base does not match compile-time description.
	DBM option not set or CLEAR/START required.
	Data base not at proper level.
	Data base not open prior to first operation.
CLOSEERROR	Data base not open.
NORECORD	Current-record pointer not valid for an INSERT operation.
	Current-record pointer not valid for a FIND manual subset-identifier operation.
	Current record of master not valid.
INUSE	Attempt made to delete a record with non-null embedded structure.
DMERROR	This attribute is set whenever any exception has occurred. One of the preceding attributes is also set.
	Note that NOT DMSTATUS (DMERROR) is true on a successful operation.

In the example below, no action is taken regardless of whether an exception occurs. The exception condition is reserved for later processing.

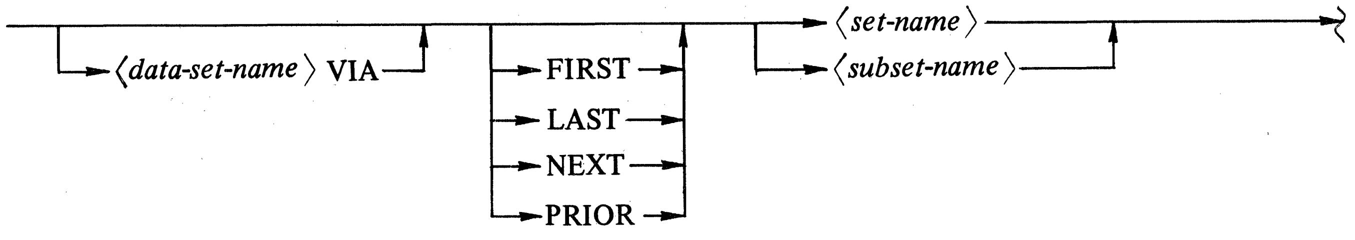
FIND MSFSET AT SSNO = C-SSNO ON EXCEPTION NEXT SENTENCE.

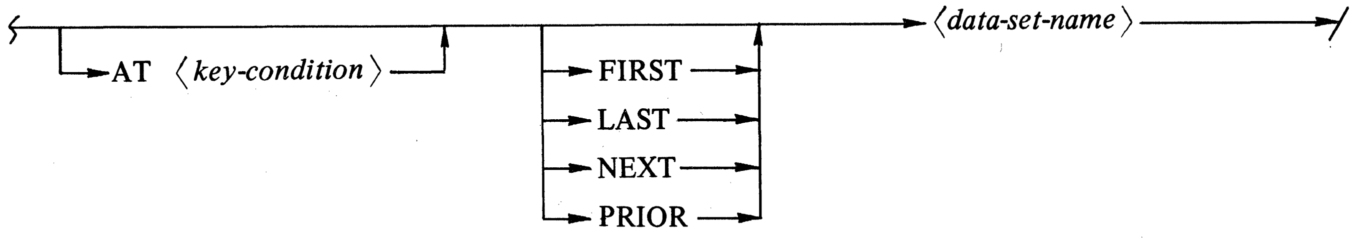
Selection Expression

The selection expression specifies the particular record of a data set that is desired, as well as the record area to be loaded with the found record. All record selections are made through paths. Paths are the routes the system uses to locate records; the physical order in which records exist in a data set constitutes a path. Any ordering keys or retrieval keys are paths, and a subset is a path.

The verbs used with selection expressions are FIND or MODIFY. Both of these verbs cause the record specified by the selection expression to be located. However, if a record which satisfies the selection expression is not found, an exception is returned. If a record is found, it is transferred into the record area. For a MODIFY operation, the found record is locked so that a concurrent user cannot access the same record. The current-record pointer is updated, and the current-path pointer for the paths is updated. Unused paths are unaffected. If a path is used and the desired record is not found, that current-path pointer becomes undefined, but the current-record pointer and record area retain the values held prior to the beginning of the operation.

Syntax





Example:

MODIFY MSFSET AT SSNO = C-SSNO
FIND MSF VIA MSFSET AT SSNO = C-SSNO
FIND MSF VIA FIRST MSFSET
FIND FIRST MSF

Semantics

a. A selection expression is used in FIND and MODIFY statements to identify a particular record in a data set.

b. The optional phrase "data-set-name VIA" at the beginning of some forms of the selection expression must be used when the path used is a manual subset.

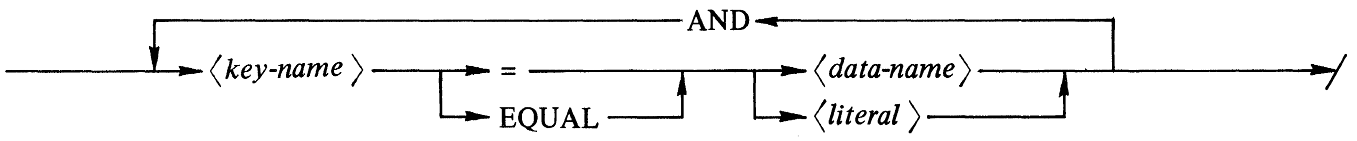
c. "data-set-name VIA" identifies the record area and current-record pointer that is affected, providing the desired record is found. By default, the data set is the data set containing the set used.

d. Note that a subset-name is interchangeable with set-name in selection expressions.

KEY CONDITION

The key condition specifies values used to locate specific records in a data set spanned by a set or referenced by a subset.

Syntax



Example:

FIND S AT A = 50 AND B = 50

Semantics

a. The key-name must be a data-name in the key as defined by the DASDL description.

b. Each key-name in the key must appear only once and to the left side of the equal sign.

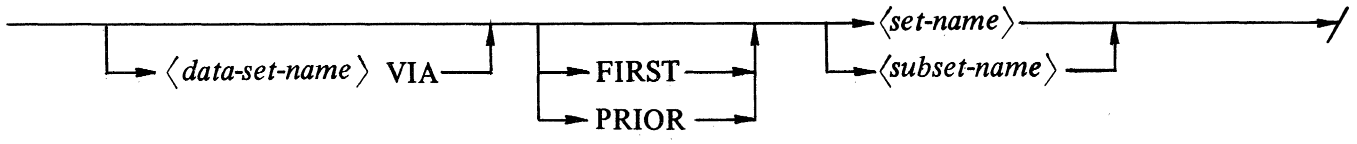
c. The valid item types for literal or data-name are determined by the COBOL MOVE statement rules. Therefore it must be legal to perform a MOVE operation on a literal or data-name to the key-name in order for the key condition to be valid.

d. The key-name of a multi-item key must appear in the same order as specified in DASDL.

SELECTION EXPRESSION FORMS

For discussion purposes, the following syntax forms are considered separately. Whenever an ordering is required but no explicit ordering exists, an implicit physical ordering is used. Whenever a current-record pointer or current-path pointer is required but is not in the proper state, the operation terminates with an exception.

Form 1:



In form 1, FIRST specifies that the first record in the specified path is to be selected. The path cannot be a retrieval set. The path is maintained in the sequence specified. If a subset is used, the data-set-name VIA clause must be used. The record returned is the first in the physical order of the subset if a key was not specified for the subset. If a key was specified for the subset, FIND data-set-name VIA FIRST subset-name locates the first record in the subset, depending on the specified ordering key.

NEXT is used to find the next record by the path specified. NEXT of a set or subset with a key returns the record with the next higher (or lower, if descending) key value; NEXT of a subset without a key locates the next physical record.

LAST locates the last record in the specified path. PRIOR locates the preceding record. NEXT and PRIOR are always relative to the current-path pointer. FIND PRIOR of a data set (form 2) can return a different record than FIND PRIOR of a set (form 1). The current-path pointer is updated to reflect the record located. NEXT and PRIOR can be used only if the current-path pointer is defined or deleted; otherwise an exception condition is returned.

Example:

```
D DATA SET
      (A NUMBER (3);
       B NUMBER (10));
       K ORDERED SET OF D KEY (A);
```

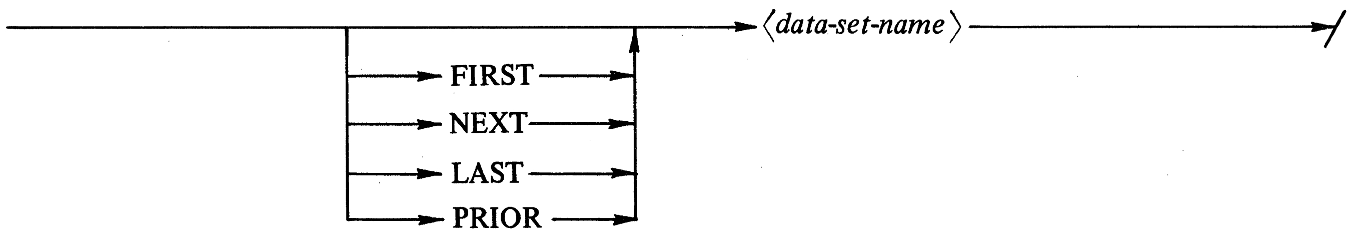
Since ascending sequence is the default ordering sequence for keys, the path K in the example below refers to members of D in sequence on A. A FIRST K therefore would transfer to the record area for D the record whose value of A was the lowest in the data set. The physical ordering of D might be different from the logical ordering represented by K. If another ordering key, K1, was added with the specification K1 ORDERED SET OF KEY (A DESCENDING), the statement FIND FIRST K1 would return the member of D with the highest value of A.

Example:

```
D DATA SET
    (A NUMBER (5);
     B ALPHA (4));
     K ORDERED SET OF D KEY (A);
D1 DATA SET
     X NUMBER (4);
     Y SUBSET OF D;
     Z SUBSET OF D KEY (B);
     Z1 ALPHA (2));
```

If D and D1 are both invoked, the statement FIND D VIA FIRST Y can then be used, returning the first physical record of D in the table of subset Y. If the statement FIND D VIA FIRST Z is used, the record found is that record of D having the lowest value of B which was inserted into Z.

Form 2:



In form 2, FIRST specifies that the record selected is the "first" physically located record in the file in which the data set is stored.

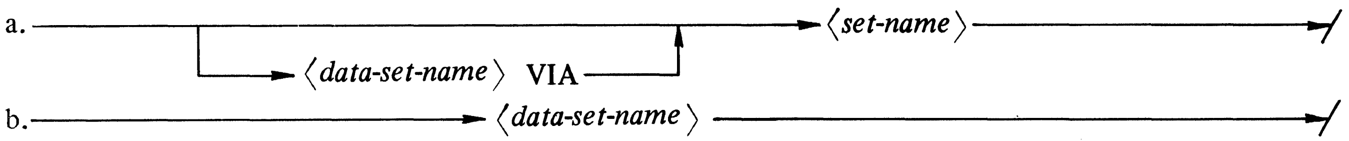
NEXT data-set-name locates the next physical record.

LAST locates the last physical record in the specified path.

PRIOR locates the preceding record.

The current-record pointer is updated to reflect the located record. NEXT and PRIOR are valid only if the current-record pointer is in the defined or deleted state; otherwise an exception condition is returned.

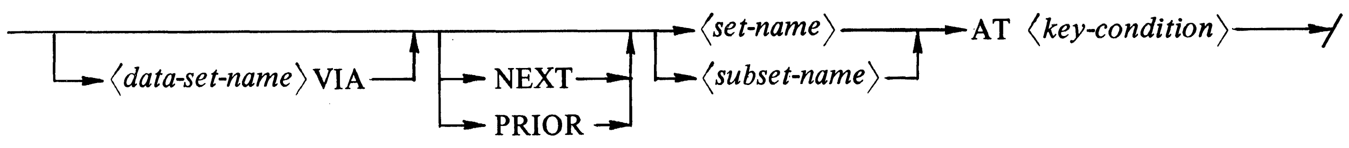
Form 3:



Form 3a of the selection expression recopies the record referenced by the current-path pointer into the record area. The current-path pointer must be in the defined state; it remains unchanged. The current-record pointer is updated.

Form 3b recopies the record referenced by the current-record pointer into the record area. The current-record pointer must be in the defined state; it remains unchanged.

Form 4:



Form 4 is used to select records of the data set based on some values in the key fields. NEXT is valid only if the current-path pointer is in the defined or deleted states.

If the NEXT expression is used, the system selects the next record in the set that satisfies the key condition. If there are no more records that satisfy the key condition, the "NOTFOUND" exception is given. Form 4 can be used only where the set-name has a key associated with it.

Example:

```
D DATA SET
    (A ALPHA (2);
     B NUMBER (10);
     C NUMBER (4));
K ORDERED SET OF D KEY (A);
K1 RETRIEVAL SET OF D KEY (C);
K2 RETRIEVAL SET OF D KEY (C,B);
```

In the above example, records of D could be selected based on the value of A, using K, or based on the value of C, using K1, or based on the values of C and B, using K2, as shown below:

```
FIND K AT A  =  "AA"
FIND K1 AT C  =  100
FIND K2 AT C  =  AND B  =  1001007890
FIND K1 AT C  =  B1
```

COBOL Statements

The COBOL verbs used to manipulate data sets are as follows:

CREATE
DELETE
FIND
FREE
INSERT
MODIFY
RECREATE
REMOVE
STORE

In addition, syntax has been implemented for the verb **OPEN**, and additional semantics for the verb **CLOSE.**

Each of the above verbs is discussed in alphabetical order in the following paragraphs.

Syntax:

$$\longrightarrow \text{CLOSE} \longrightarrow \langle \textit{data-base-name} \rangle \longrightarrow /\!/$$

Semantics:

a. **CLOSE** can be used to close a data base when further access is no longer required.

b. **CLOSE** is optional, since the system closes any open data base when the program terminates.

c. An implicit **FREE** is performed on all records locked by the program.

d. If the data base is not open, the operation terminates with an exception condition.

CREATE

Syntax:

```
CREATE ——————————————→ ⟨data-set-name⟩ ——————————————————→//
```

Semantics:

a. **CREATE** must be performed prior to the addition of a new record in a data set, (optionally **RECREATE** may be used). A **CREATE** does not add the new record to the data base; that is the function of a **STORE**. The main purpose of a **CREATE** is to initialize the entire current record area of the data set to null (all bits = 1). This is used for validity checking of the record at the time of the **STORE** operation.

b. An implicit **FREE** is performed on the prior current record of the data set.

c. The current-record pointer goes to the created state.

d. Normally, **CREATE** is eventually followed by a **STORE**, placing the new record into the data set. However, if a subsequent **STORE** is not desired, the **CREATE** can be nullified by a subsequent **FIND, MODIFY, CREATE,** or **RECREATE.**

e. A **CREATE** initializes only a record area. If the record contains embedded structures, the master record must be stored before storing entries in the embedded structure. If only entries in the embedded structure are added, changed, or deleted the master need not be stored a second time.

Syntax:

```
————————► DELETE ——————————► ⟨data-set-name⟩ ——————————————————————►//
```

Semantics:

a. The **DELETE** operation eliminates a specified record from a data set.

b. The current record area is reloaded with the contents of the record.

c. If the record contains a non-empty embedded structure, the record is not deleted.

d. If the record can be deleted, it is removed from all sets and automatic subsets of which it is a member. The record is then removed from the data set. The current-record pointer goes to the deleted state. The data remains unaltered in the record area.

e. The user must remove the record from any manual subset that points at the data set record being deleted (refer to the **REMOVE** statement).

FIND

Syntax:

$$\longrightarrow \text{FIND} \longrightarrow \langle \textit{selection-expression} \rangle \longrightarrow /\!/$$

Semantics:

a. The **FIND** operation performs two functions:

   1. Locates the record satisfying the selection-expression and

   2. Transfers the data from the data base to the record area so it can be accessed by the program.

b. If a record satisfying the selection-expression is not found, the operation terminates with an exception condition. In this case, the record area and current-record pointer retain their original values. However, if a set or subset had been involved, its current-path pointer becomes undefined.

c. If a record is found, it is transferred to the record area, and the current-record pointer is altered to refer to the found record. Also, if a set or automatic subset had been involved, its current-path pointer is altered to refer to the found record.

d. Prior to the **FIND** operation, an implicit **FREE** is performed to unlock the previous current record.

Syntax:

```
──────────►FREE──────────────────► <data-set-name> ──────────────────────────►//
```

Semantics

a. A **FREE** operation unlocks the current record.

b. A **FREE** can occur after any operation. If the current-record pointer is not in the defined state or the current record is not locked, the **FREE** is ignored.

c. A **FREE** is optional in most situations, since the **CREATE**, **RECREATE** (and sometimes the **FIND** or **MODIFY** operation) perform an implicit FREE prior to their other actions. In general, an implicit FREE is performed prior to any operation that establishes a new current-record pointer.

d. The current-record pointer and current-record area are not affected.

INSERT

Syntax:

```
INSERT ———► ⟨data-set-name⟩ ———► INTO ———► ⟨manual-subset-name⟩ ———————►//
```

Semantics:

a. The **INSERT** operation is used to insert a record into a manual subset.

b. The data-set-name must be the declared source of records for a manual subset. For example, the manual subset-name must be a manual subset of data-set-name, as the example below illustrates.

   DASDL: S1 SUBSET OF D
   COBOL: INSERT D INTO S1

c. The current-record pointer of data-set-name must be defined; if not, the operation is terminated with an exception condition.

d. The data set in which the manual subset is embedded must have the current-record pointer in the defined state, and that record must be locked; if not, the operation is terminated with an exception condition.

e. If duplicates are not allowed for the manual subset, an exception condition occurs if a record that has a key identical to that of the source record already exists in the manual subset.

MODIFY

Syntax:

$$\begin{tikzcd} {} \arrow[r] & \text{MODIFY} \arrow[r] & \langle \textit{selection-expression} \rangle \arrow[r] & // \end{tikzcd}$$

Semantics:

a. The functions of a **MODIFY** operation are identical to **FIND** with one exception: if the record is found, it is locked, prohibiting concurrent modification by another user.

b. A **MODIFY** operation should be used if there is a possibility that the data set record contents will be changed. The **MODIFY** operation does not physically modify the record, but allows modification to be performed subsequently without a concurrent update from another user.

c. If the found record is already locked by another user, a contention analysis is performed by the system. Normally, the present user waits until the record is unlocked. However, if it is determined that waiting would result in a "deadly embrace," all records locked by the present user are unlocked; and the operation is terminated with an exception condition.

d. Since no other user may lock a record once it is locked, it is important to free the record when it is no longer necessary to keep it locked. This is accomplished by a **FREE** operation or implicitly by a subsequent **MODIFY**, **FIND**, **CREATE**, or **RECREATE** on the same data set. A subsequent **STORE** leaves the record locked.

e. The locking action is maintained on a block level.

OPEN

Syntax:

```
────────► OPEN ──────────► UPDATE ──────────►〈data-base-name〉────────────────►//
```

Semantics:

a. The **OPEN** operation is used to open a data base for subsequent access.

b. An **OPEN** must be executed prior to the first access to the data base; otherwise, all data base requests will terminate with an exception condition.

c. If the data base is already open, the operation is terminated with an exception condition.

d. The system attempts to open an existing data base. The data base dictionary is opened at this time. If the data base dictionary is not present, the message:

   NO FILE *data-base-name* /**DICTIONARY**

   is displayed.

   Each data file is opened on the first operation that uses the data in the file. Files that are not needed are not opened. If the files are needed, and they are not present when they are needed, the message:

   NO FILE *file-identifier*

   is displayed.

Syntax:

```
——————► RECREATE ——————► <data-set-name> ——————————————►//
```

Semantics:

RECREATE operation is identical to **CREATE**, with one exception: the record area for the data set is not completely initialized. All data items remain unaltered; however, items such as manual subsets and embedded data sets are set to null.

REMOVE

Syntax:

```
————▶ REMOVE ——▶ CURRENT ——▶ FROM ——▶ ⟨manual-subset-name⟩ ——▶//
```

Semantics:

a. The **REMOVE** operation is used to remove a record from a manual subset.

b. The manual subset must have a defined current-path pointer; if not, the operation is terminated with an exception condition.

c. The record referenced by the manul subset current-path pointer is removed from the subset but not from the data set.

d. The data set in which the manual subset is embedded must have the current-record pointer in the defined state and that record must be locked; if not, the operation is terminated with an exception condition.

Syntax:

```
──────► STORE ──────────────► <data-set-name> ──────────────────►//
```

Semantics:

a. The **STORE** operation is used to return a modified record to a data set, or to place a newly created record into a data set.

b. The data to be stored is in the record area of the data set. Prior to the storing of a record, the data is checked for validity (VERIFY, REQUIRED, non-null keys) as specified by DASDL. A validity failure terminates the **STORE** operation with an exception condition.

c. If the current-record pointer is in the defined state and the current record is locked, the data replaces the current record in the data set remains locked. If the current-record pointer is in a defined state but unlocked, or in an undefined state or deleted state, the operation terminates with an exception condition.

d. If the current-record pointer is in the created state, the data becomes a new record in the data set and is locked. The current-record pointer is then in the defined state and refers to the new record.

e. Set current-path-pointer is not affected by a **STORE** operation.

f. All fields which are, or form, part of a key or are REQUIRED must contain a value other than a null value before a **STORE** operation can be completed successfully. If any of these fields are null, the operation terminates with an exception condition.

g. The following additional actions are performed depending on the prior operation.

STORE After **CREATE** or **RECREATE**

1. The condition is evaluated for each automatic subset (subset containing a WHERE condition). The subset is marked for insertion if the condition and validity checks are satisfied.

2. If a data record cannot be inserted into any set (but not automatic subsets) for some reason, the operation is terminated with an exception condition. In this case, the record is not inserted into the data set nor is it inserted into any set. If no reason does exist, the STORE operation is successful and all necessary set and automatic subset insertions are made.

3. For each set that spans the data set, the record is tested for validity. After the STORE operation, any data item not containing a value will be null (all bits = 1). Care should be taken in the **COBOL** program if it is necessary to check the contents of such a field, because the contents will be hexadecimal F's.

STORE After **MODIFY**

1. In this operation, the record already exists in all sets.

2. Conditions must be re-evaluated if any items involved in the automatic subsets condition have changed. The record is removed from the automatic subsets containing the record if a condition is not satisfied. The record is inserted into automatic subsets not already containing the record if the condition is satisfied.

3. If a key used in the ordering of a set is modified, and the record must be moved in that set, the record is deleted from the set and reinserted in the proper position. It is illegal to modify a key if duplicates are not allowed, or if the set is an embedded set.

4. If the ordering of a manual subset is affected, the **STORE** operation will occur, but no reordering of that manual subset will be performed. It is the responsibility of the user to maintain manual subsets. A subsequent reference to the record through that subset causes the operation to be terminated with an exception condition (DMSTATUS (KEYCHANGED).

Appendix A. DMSII COMPILATION GUIDE

DASDL COMPILATION PROCEDURES

The following procedures must be observed to create a data base:

a. Compilation of a DASDL source deck defines the logical and, optionally, the physical structure of the data to be entered into the data base. The DASDL compilation types are as follows:

   1. Compile for syntax: Causes a syntax check of the DASDL input. Neither a dictionary file nor library files are generated.

   2. Compile for library: Causes a data base dictionary file to be created and removes any existing dictionary file having the same name.

      NOTE

      The compile-and-go operation is not recommended for use, as it implies the production of object code. Since there is no object code produced, the MCP displays a warning message.

b. The data base dictionary created by a successful compilation is titled:

   ⟨data-base-identifier⟩/**DICTIONARY**

   The data-base-identifier is the "family-name" of the program identifier used for the DASDL compilation.

   The data base dictionary file is a disk file containing a description of all the structural characteristics of the data base. This description is used during execution by the MCP access routines to control all access to the data base.

c. The INITIALIZE statement is required prior to the execution of any program accessing the data base. Its function is to create initialized disk files for storing data records and index tables. This function is not performed on a compile for syntax, or a compilation where there are syntax errors. Once INITIALIZE has been executed, a permanent data base is created. No further changes can be made without recreating the data base by a recompilation of the DASDL source deck.

d. The following examples provide illustrations of a DASDL compilation source deck.

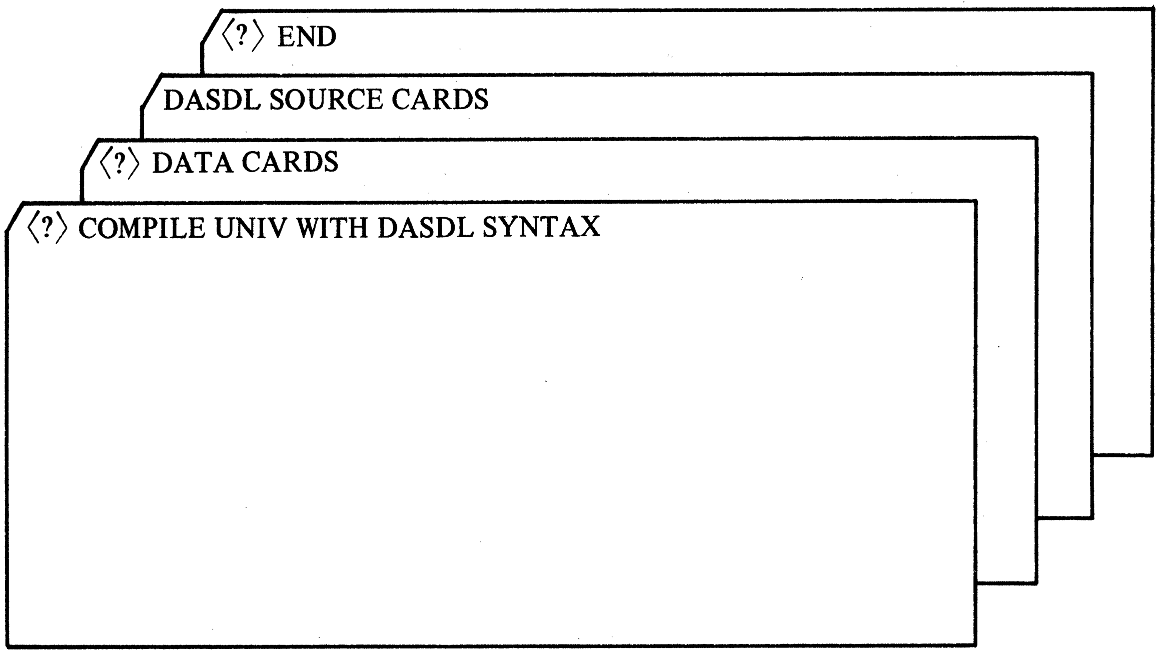


Figure A-1. Compile for Syntax

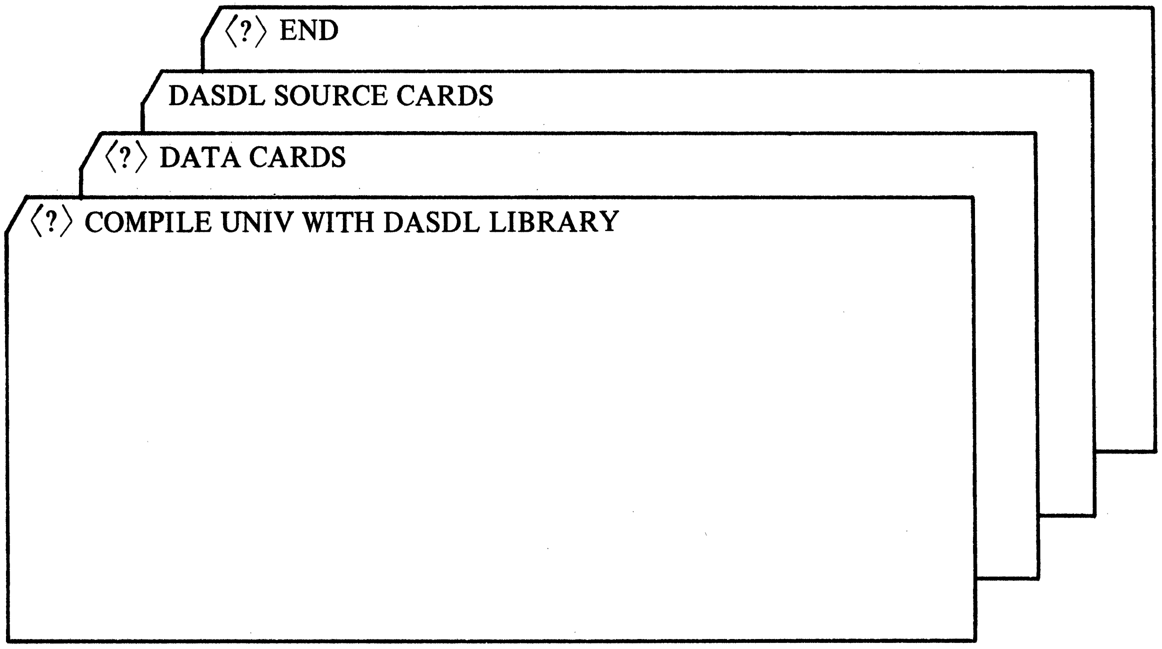


Figure A-2. Compile for Library

DASDL COMPILATION PROGRAM NAME

A program name consists of either one, two, or three identifiers, with each identifier able to be a maximum of 10 characters in length. The following example illustrates the four possible program name formats:

```
                        <family-name>
                        <family-name>/<file-identifier>
<disk-pack-identifier>/<family-name>/
<disk-pack-identifier>/<family-name>/<file-identifier>
```

The family-name is synonymous with the data-base-name. For further information regarding program name formulation, refer to the B 1700 System Software Operational Guide, Form No. 1068731.

DASDL FILE NAMES

The following file names are produced by a successful DASDL compilation.

a. Dictionary file: *⟨data-base-name⟩* /DICTIONARY

b. COBOL library file: *⟨data-base-name⟩* / *⟨disjoint-data-set-name⟩*

c. Data files:

Data files:	*⟨data-base-name⟩*/*⟨data-set-name⟩*	(Data sets)
	⟨data-base-name⟩/*⟨subset-name⟩*	(Subsets)
	⟨data-base-name⟩/*⟨set-name⟩*	(Disjoint sets)

If a set is stored in a file with a data set, the data-set naming convention takes precedence. If multiple sets are stored together, the name of the first set has precedence. Embedded sets are stored with their data sets.

If the program name of a DASDL compilation has a disk-pack-id, all data files will have that disk-pack-id appended to them. Both the dictionary file, at program execution, and the COBOL library file, at compilation time, are expected to reside on system disk.

DASDL COMPILER ($) OPTIONS

The following compiler options can appear either at the beginning or within the source deck. The format of the $ option card is below.

```
$    [NO]    option-1    [NO]    option-2    . . .
```

Option	Description
$ COBOL	Checks for COBOL reserved words, and must appear before any source cards.
$ DOUBLE	Causes listing to be double spaced.
$ FILE	Causes the printing of the file attributes.
$ LIST	Causes a printout of the listing. Automatically set by compiler.
$ NO	Negates the option directly following the word NO.
$ SINGLE	Causes the listing to be single spaced.
$ SOURCE	Causes the generated COBOL library files to be printed. The SOURCE option is usually placed prior to the source deck physical description.
$ SOURCEONLY	Regenerates the COBOL copy files by using an existing data base dictionary. This option is the only input for this DASDL run.
$ STRUCTURE	Causes the printing of structure attributes. If STRUCTURE is placed before the physical description, both the default and revised structure attributes will be printed. If it is placed after the first physical attribute, only the revised structure attributes will be printed.
$ SUPPRESS	Causes warning messages to be suppressed.
$ VERSIONCHECK	Automatically set by the compiler. When the option is in a reset condition ($ NO VERSIONCHECK) the compiler provides the mechanism to ignore the program dictionary checking performed at execution. This action eliminates the requirement of program recompilation with each new DASDL recompilation.

NOTE

$ VERSIONCHECK should be set for all non-testing executions.

Figure A-3 contains a DASDL program with the $ FILE and $ STRUCTURE options specified. The $ STRUCTURE option causes each structure to be printed with all of its parameters. The $ FILE option causes the file parameters to be printed following the structures with which it contains. Thus, UNIV-PERSONNEL and SS-U-P are contained in file number 1, named UNIV/UNIV-PERSO.

```
: UNIV       DATA BASE
:$LIST SINGLE

:$SUPPRESS
:$FILE STRUCTURE
:%THIS DASDL PROGRAM GIVES EXAMPLES
:%OF THE VARIOUS CONSTRUCTS USED IN
:%DASDL TO DESCRIBE A DATA BASE
:PARAMFTERS(
:     BUFFERS = 10    );
:UNIV-COURSES DATA SET "MAIN FILE" (
:     CRS-NAME GROUP (
:          DEPARTMENT ALPHA(2);
:          LEVEL NUMBER(3);
:          CRS-NO NUMBER(4));
:     NOPROF NUMBER(2);
:     DAYS-OF-WEEK GROUP (
:          MON NUMBER(1);
:          TUES NUMBER(1);
:          WEDS NUMBER(1);
:          THURS NUMBER(1);
:          FRI NUMBER(1);
:          SAT NUMBER(1));
:     BUILDING NUMBER(3);
:     ROOMNUMBER ALPHA(2);
:     COURSENAME ALPHA(24);
:     FLAG-BITS ALPHA(12);
:     HOURSCRDT NUMBER(4);
:     CLASS-SIZE NUMBER(2);
:     PROFESSOR SUBSET OF UNIV-PERSONNEL,POPULATION = 3;
:     BOOKS UNORDERED DATA SET(
:          LC NUMBER(9);
:          TITLES ALPHA(60);
:          AUTHR ALPHA(30));
:     STUDENTS SUBSET OF MSF KEY IS
:          (LNAME,FNAME)DUPLICATES,
:          POPULATION = 300)
:     POPULATION = 1000
:VERIFY((HOURSCRDT GTR 0 AND CLASS-SIZE LEQ 60) AND NOPROF NEQ 0);
:     UNIV-C-SET ORDERED SET OF UNIV-COURSES KEY IS (CRS-NO);
:UNIV-PERSONNEL DATA SET(
:     NAME GROUP(
:          LASTNAME ALPHA(15);
:          FIRSTNAME ALPHA(10));
:     SEX NUMBER(1);
:     AGE NUMBER(2);
:     SSNUM NUMBER(9);
:     DPT ALPHA(4);
:     RANK ALPHA(1);
:     SALARY NUMBER(S7,2);
:     COURSES SUBSET OF UNIV-COURSES,POPULATION = 8;
:     ADDRES SUBSET OF ADR;
:     SUPR SUBSET OF UNIV-PERSONNEL);
:     SS-U-P ORDERED SET OF UNIV-PERSONNEL KEY IS (SSNUM);
:     U-P-SET ORDERED SET OF UNIV-PERSONNEL KEY IS
:          (LASTNAME,FIRSTNAME) DUPLICATES;
:MSF DATA SET(
:     SSNO NUMBER(9);
:     NONAM NUMBER(1);
:     LNAME ALPHA(30);
:     MNAME ALPHA(30);
```

Figure A–3. UNIV Data Base Example (Sheet 1)

```
00005900 :    FNAME ALPHA(30);                                                    :
00006000 :    CAMPUS-ADDRESS GROUP(                                               :
00006100 :         DORM ALPHA(6);                                                 :
00006200 :         ROOM NUMBER(4);                                                :
00006300 :         POBOX NUMBER(4);                                               :
00006400 :         PHONE NUMBER(7));                                              :
00006500 :    NO NUMBER(2);                                                       :
00006600 :    DEGREE ALPHA(4) OCCURS 6 TIMES;                                     :
00006700 :    TOTHRS NUMBER(3);                                                   :
00006800 :    TOTQP NUMBER(3);                                                    :
00006900 :    GRADE-POINT-AVG NUMBER(3,2);                                        :
00007000 :    MJR NUMBER(3);                                                      :
00007100 :    AMJR ALPHA(18);                                                     :
00007200 :    SSEX NUMBER(1);                                                     :
00007300 :    SAGE NUMBER(2);                                                     :
00007400 :    HOME-ADDRESS SUBSET OF ADR;                                         :
00007500 :    QUARTER   ORDERED DATA SET(                                         :
00007600 :         QTR ALPHA(4);                                                  :
00007700 :         QTTRHRS NUMBER(2);                                             :
00007800 :         QTRQP NUMBER(2);                                               :
00007900 :         CORSES ORDERED DATA SET(                                       :
00008000 :              TYPECOURSE NUMBER(1) REQUIRED;                            :
00008033 :              GRADE ALPHA(2) REQUIRED;                                  :
00008066 :              CRS SUBSET OF UNIV-COURSES;                               :
00008100 :              YR NUMBER(2);                                             :
00008200 :              Q NUMBER(2);                                              :
00008300 :              GCRS SUBSET OF UNIV-COURSES;                              :
00008400 :              GGD ALPHA(2);                                             :
00008500 :              TITLE-OF-PAPER ALPHA(30);                                 :
00008600 :              PPRGD ALPHA(2)),                                          :
00008700 :              POPULATION = 4;                                           :
00008800 :              CSET ACCESS TO CORSES KEY IS                              :
00008850 :                   (TYPECOURSE) DUPLICATES)                             :
00009000 :         POPULATION = 5000;                                             :
00009100 :         QSET ACCESS TO QUARTER KEY IS (QTR));                          :
00009200 :    MSFSET ORDERED SET OF MSF KEY IS (SSNO);                            :
00009300 :ADR DATA SET(                                                           :
00009400 :    FACULTY-STUDENT NUMBER(1);                                          :
00009500 :    SNO NUMBER(9) REQUIRED;                                             :
00009600 :    ADLN ALPHA(54) OCCURS 9 TIMES;                                      :
00009700 :    ZIPC NUMBER(5) REQUIRED;                                            :
00009800 :    PHON NUMBER(10));                                                   :
00009850 :SAD ORDERED SET OF ADR KEY IS (ZIPC) DUPLICATES;                        :
00009900 :    SSAD ORDERED SET OF ADR KEY IS (SNO);                               :
00009910 :STUAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 1) KEY IS (ZIPC,SNO)    :
00009920 :                                                      DUPLICATES;       :
00009930 :FACAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 2) KEY IS (ZIPC,SNO)    :
00009940 :                                                      DUPLICATES;       :
00009950 :ADMAD SUBSET OF ADR WHERE(FACULTY-STUDENT EQL 3) KEY IS (ZIPC,SNO)     :
00009960 :                                                      DUPLICATES;       :
00010500 :BOOKS(                                                                  :
```

```
UNIV-COURSES DATA SET :
STRUCTURE NUMBER    1
PRIME = 0
BLOCKSIZE =      2 RECORDS/BLOCK
AREASIZE =   292 RECORDS PER AREA
POPULATION =     1000 RECORDS
DATASIZE =   424 BITS
RECORDSIZE =   616 BITS
```

Figure A–3. UNIV Data Base Example (Sheet 2)

```
BITS PER BLOCK =  1232


UNIV-C-SET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER   7
PRIME = 1
TABLESIZE(COARSE) =     70 ENTRIES
TABLESIZE(FINE) =     58 ENTRIES
AREASIZE =      7 TABLES PER AREA
LOADFACTOR =     65 PERCENT
SPLITFACTOR = 50 PERCENT
ENTRYSIZE(COARSE) =     40 BITS
ENTRYSIZE(FINE) =     48 BITS
BITS PER BLOCK(COARSE) =  2816
BITS PER BLOCK(FINE) =  2800


PROFESSOR UNORDERED LIST :
STRUCTURE NUMBER   2
AREASIZE =     50 RECORDS PER AREA
TABLESIZE =     42 ENTRIES
BLOCKSIZE =      8 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =     32 BITS
BITS PER BLOCK = 11360


STUDENTS ORDERED LIST :
STRUCTURE NUMBER   5
AREASIZE =  7500 RECORDS PER AREA
TABLESIZE =      2 ENTRIES
BLOCKSIZE =      1 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1096 BITS
ENTRYSIZE =   512 BITS
BITS PER BLOCK =  1128


BOOKS UNORDERED LIST :
STRUCTURE NUMBER   4
AREASIZE =   500 RECORDS PER AREA
TABLESIZE =      1 ENTRIES
BLOCKSIZE =      1 RECORDS/BLOCK
DATASIZE =   760 BITS
RECORDSIZE =    832 BITS
ENTRYSIZE =    760 BITS
BITS PER BLOCK =    864


UNIV-PERSONNEL DATA SET :
STRUCTURE NUMBER   3
PRIME = 0
BLOCKSIZE =      2 RECORDS/BLOCK
AREASIZE =  1192 RECORDS PER AREA
POPULATION =    10000 RECORDS
DATASIZE =   320 BITS
RECORDSIZE =    512 BITS
BITS PER BLOCK =  1024
```

Figure A–3. UNIV Data Base Example (Sheet 3)

```
SS-U-P INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  12
PRIME = 1
TABLESIZE(COARSE) =   143 ENTRIES
TABLESIZE(FINE) =   119 ENTRIES
AREASIZE =    12 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    60 BITS
ENTRYSIZE(FINE) =    72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


U-P-SET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  13
PRIME = 0
TABLESIZE(COARSE) =   109 ENTRIES
TABLESIZE(FINE) =   105 ENTRIES
AREASIZE =    13 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =   224 BITS
ENTRYSIZE(FINE) =   232 BITS
BITS PER BLOCK(COARSE) = 24432
BITS PER BLOCK(FINE) = 24376


COURSES UNORDERED LIST :
STRUCTURE NUMBER   8
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


ADDRES UNORDERED LIST :
STRUCTURE NUMBER   9
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


SUPR UNORDERED LIST :
STRUCTURE NUMBER  11
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696
```

Figure A–3. UNIV Data Base Example (Sheet 4)

```
MSF DATA SET :
STRUCTURE NUMBER    6
PRIME = 0
BLOCKSIZE =      1 RECORDS/BLOCK
AREASIZE =  1191 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  1276 BITS
RECORDSIZE =  1404 BITS
BITS PER BLOCK =  1404


MSFSET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  19
PRIME = 1
TABLESIZE(COARSE) =   143 ENTRIES
TABLESIZE(FINE) =   119 ENTRIES
AREASIZE =    12 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    60 BITS
ENTRYSIZE(FINE) =    72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


HOME-ADDRESS UNORDERED LIST :
STRUCTURE NUMBER  14
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


CRS UNORDERED LIST :
STRUCTURE NUMBER  17
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


GCRS UNORDERED LIST :
STRUCTURE NUMBER  18
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


CORSES ORDERED LIST :
STRUCTURE NUMBER  16
```

Figure A–3. UNIV Data Base Example (Sheet 5)

```
AREASIZE = 14339 RECORDS PER AREA
TABLESIZE =      3 ENTRIES
BLOCKSIZE =      1 RECORDS/BLOCK
DATASIZE =   312 BITS
RECORDSIZE =  1392 BITS
ENTRYSIZE =   440 BITS
BITS PER BLOCK =  1424


QUARTER ORDERED LIST :
STRUCTURE NUMBER  15
AREASIZE = 20850 RECORDS PER AREA
TABLESIZE =     12 ENTRIES
BLOCKSIZE =      1 RECORDS/BLOCK
DATASIZE =    48 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =   112 BITS
BITS PER BLOCK =  1448


ADR DATA SET :
STRUCTURE NUMBER  10
PRIME = 0
BLOCKSIZE =      1 RECORDS/BLOCK
AREASIZE =   910 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  3988 BITS
RECORDSIZE =  3988 BITS
BITS PER BLOCK =  3988


SAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  20
PRIME = 1
TABLESIZE(COARSE) =   129 ENTRIES
TABLESIZE(FINE) =   101 ENTRIES
AREASIZE =    11 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    44 BITS
ENTRYSIZE(FINE) =    56 BITS
BITS PER BLOCK(COARSE) =  5692
BITS PER BLOCK(FINE) =  5672


SSAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  21
PRIME = 0
TABLESIZE(COARSE) =   143 ENTRIES
TABLESIZE(FINE) =   119 ENTRIES
AREASIZE =     9 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    60 BITS
ENTRYSIZE(FINE) =    72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


STUAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  22
```

Figure A–3. UNIV Data Base Example (Sheet 6)

```
                                        PRIME = 0
                                        TABLESIZE(COARSE) =   125 ENTRIES
                                        TABLESIZE(FINE) =   113 ENTRIES
                                        AREASIZE =    10 TABLES PER AREA
                                        LOADFACTOR =    66 PERCENT
                                        SPLITFACTOR = 49 PERCENT
                                        ENTRYSIZE(COARSE) =    80 BITS
                                        ENTRYSIZE(FINE) =    88 BITS
                                        BITS PER BLOCK(COAPSE) = 10016
                                        BITS PER BLOCK(FINE) =  9960


                                        FACAD INDEX SEQUENTIAL SET !
                                        STRUCTURE NUMBER  23
                                        PRIME = 0
                                        TABLESIZE(COARSE) =   125 ENTRIES
                                        TABLESIZE(FINE) =   113 ENTRIES
                                        AREASIZE =    10 TABLES PER AREA
                                        LOADFACTOR =    66 PERCENT
                                        SPLITFACTOR = 49 PERCENT
                                        ENTRYSIZE(COARSE) =    80 BITS
                                        ENTRYSIZE(FINE) =    88 BITS
                                        BITS PER BLOCK(COARSE) = 10016
                                        BITS PER BLOCK(FINE) =  9960


                                        ADMAD INDEX SEQUENTIAL SET !
                                        STRUCTURE NUMBER  24
                                        PRIME = 0
                                        TABLESIZE(COARSE) =   125 ENTRIES
                                        TABLESIZE(FINE) =   113 ENTRIES
                                        AREASIZE =    10 TABLES PER AREA
                                        LOADFACTOR =    66 PERCENT
                                        SPLITFACTOR = 49 PERCENT
                                        ENTRYSIZE(COARSE) =    80 BITS
                                        ENTRYSIZE(FINE) =    88 BITS
                                        BITS PER BLOCK(COARSE) = 10016
                                        BITS PER BLOCK(FINE) =  9960
00010600 !    AREASIZE = 500,
00010650 !    TYPE = UNORDERED LIST,
00010700 !    BLOCKSIZE = 5)!
00010800 !BOOKFILE STORAGE FOR BOOKS(
00010850 !    TITLE = UNIV/LIBRARY,
00010900 !    AREAS = 10)!
00011000 !UNIV-C-SET(
00011100 !    TABLESIZE = 12,
00011150 !    AREASIZE = 10,
00011200 !    TYPE = INDEX SEQUENTIAL,
00011300 !    LOADFACTOR = 9)!
00011400 !UNIV-PERSONNEL(
00011450 !    PRIME,
00011500 !    POPULATION = 997)!
00011600 !INITIALIZE!


                                        UNIV-COURSES DATA SET !
                                        STRUCTURE NUMBER   1
                                        PRIME = 0
                                        BLOCKSIZE =      2 RECORDS/BLOCK
                                        AREASIZE =   292 RECORDS PER AREA
                                        POPULATION =     1000 RECORDS
```

Figure A–3. UNIV Data Base Example (Sheet 7)

```
DATASIZE =   424 BITS
RECORDSIZE =   616 BITS
BITS PER BLOCK =  1232


UNIV-C-SET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER   7
PRIME = 1
TABLESIZE(COARSE) =    14 ENTRIES
TABLESIZE(FINE) =    12 ENTRIES
AREASIZE =    10 TABLES PER AREA
LOADFACTOR =     8 PERCENT
SPLITFACTOR = 50 PERCENT
ENTRYSIZE(COARSE) =    40 BITS
ENTRYSIZE(FINE) =    48 BITS
BITS PER BLOCK(COARSE) =   576
BITS PER BLOCK(FINE) =   592


FILE INFORMATION :
FILE NUMBER   2
AREAS =  20
AREASIZE =   156 SEGMENTS
PACK =
TITLE = UNIV      /UNIV-COURS


PROFESSOR UNORDERED LIST :
STRUCTURE NUMBER   2
AREASIZE =    50 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     8 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK = 11360


FILE INFORMATION :
FILE NUMBER   3
AREAS =  20
AREASIZE =    50 SEGMENTS
PACK =
TITLE = UNIV      /PROFESSOR


STUDENTS ORDERED LIST :
STRUCTURE NUMBER   5
AREASIZE =  7500 RECORDS PER AREA
TABLESIZE =     2 ENTRIES
BLOCKSIZE =     1 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1096 BITS
ENTRYSIZE =   512 BITS
BITS PER BLOCK =  1128


FILE INFORMATION :
FILE NUMBER   4
AREAS =  20
AREASIZE =  7500 SEGMENTS
```

Figure A–3. UNIV Data Base Example (Sheet 8)

```
PACK =
TITLE = UNIV      /STUDENTS


UNIV-PERSONNEL DATA SET :
STRUCTURE NUMBER   3
PRIME = 1
BLOCKSIZE =      2 RECORDS/BLOCK
AREASIZE =  1192 RECORDS PER AREA
POPULATION =     997 RECORDS
DATASIZE =   320 BITS
RECORDSIZE =   512 BITS
BITS PER BLOCK =  1024


FILE INFORMATION :
FILE NUMBER   5
AREAS =  20
AREASIZE =   596 SEGMENTS
PACK =
TITLE = UNIV      /UNIV-PERSO


COURSES UNORDERED LIST :
STRUCTURE NUMBER   8
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   6
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /COURSES


ADDRES UNORDERED LIST :
STRUCTURE NUMBER   9
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   7
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /ADDRES
```

Figure A–3. UNIV Data Base Example (Sheet 9)

```
SUPR UNORDERED LIST :
STRUCTURE NUMBER  11
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   8
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /SUPR


MSF DATA SET :
STRUCTURE NUMBER   6
PRIME = 0
BLOCKSIZE =      1 RECORDS/BLOCK
AREASIZE =  1191 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  1276 BITS
RECORDSIZE =  1404 BITS
BITS PER BLOCK =  1404


MSFSET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  19
PRIME = 1
TABLESIZE(COARSE) =    143 ENTRIES
TABLESIZE(FINE) =    119 ENTRIES
AREASIZE =     12 TABLES PER AREA
LOADFACTOR =     66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =     60 BITS
ENTRYSIZE(FINE) =     72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


FILE INFORMATION :
FILE NUMBER   9
AREAS =  20
AREASIZE =  1263 SEGMENTS
PACK =
TITLE = UNIV      /MSF


HOME-ADDRESS UNORDERED LIST :
STRUCTURE NUMBER  14
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696
```

Figure A–3. UNIV Data Base Example (Sheet 10)

```
FILE INFORMATION :
FILE NUMBER  10
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /HOME-ADDRE


CRS UNORDERED LIST :
STRUCTURE NUMBER  17
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =     42 ENTRIES
BLOCKSIZE =      4 RECORDS/BLOCK
DATASIZE =     32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =     32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER  11
AREAS =  20
AREASIZE =  9632 SEGMENTS
PACK =
TITLE = UNIV      /CRS


GCRS UNORDERED LIST :
STRUCTURE NUMBER  18
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =     42 ENTRIES
BLOCKSIZE =      4 RECORDS/BLOCK
DATASIZE =     32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =     32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER  12
AREAS =  20
AREASIZE =  9632 SEGMENTS
PACK =
TITLE = UNIV      /GCRS


CORSES ORDERED LIST :
STRUCTURE NUMBER  16
AREASIZE = 14339 RECORDS PER AREA
TABLESIZE =      3 ENTRIES
BLOCKSIZE =      1 RECORDS/BLOCK
DATASIZE =   312 BITS
RECORDSIZE =  1392 BITS
ENTRYSIZE =   440 BITS
BITS PER BLOCK =  1424


FILE INFORMATION :
FILE NUMBER  13
```

Figure A–3. UNIV Data Base Example (Sheet 11)

```
AREAS = 20
AREASIZE = 14339 SEGMENTS
PACK =
TITLE = UNIV      /CORSES


QUARTER ORDERED LIST :
STRUCTURE NUMBER  15
AREASIZE = 20850 RECORDS PER AREA
TABLESIZE =    12 ENTRIES
BLOCKSIZE =     1 RECORDS/BLOCK
DATASIZE =    48 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =   112 BITS
BITS PER BLOCK =  1448


FILE INFORMATION :
FILE NUMBER  14
AREAS =  20
AREASIZE = 41700 SEGMENTS
PACK =
TITLE = UNIV      /QUARTER


ADR DATA SET :
STRUCTURE NUMBER  10
PRIME = 0
BLOCKSIZE =     1 RECORDS/BLOCK
AREASIZE =   910 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  3988 BITS
RECORDSIZE =  3988 BITS
BITS PER BLOCK =  3988


SAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  20
PRIME = 1
TABLESIZE(COARSE) =   129 ENTRIES
TABLESIZE(FINE) =   101 ENTRIES
AREASIZE =    11 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    44 BITS
ENTRYSIZE(FINE) =    56 BITS
BITS PER BLOCK(COARSE) =  5692
BITS PER BLOCK(FINE) =  5672


FILE INFORMATION :
FILE NUMBER  15
AREAS =  20
AREASIZE =  2774 SEGMENTS
PACK =
TITLE = UNIV      /ADR


SSAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  21
PRIME = 0
```

Figure A–3. UNIV Data Base Example (Sheet 12)

```
TABLESIZE(COARSE) =   143 ENTRIES
TABLESIZE(FINE) =   119 ENTRIES
AREASIZE =      9 TABLES PER AREA
LOADFACTOR =     66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =     60 BITS
ENTRYSIZE(FINE) =     72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =     54 SEGMENTS
PACK =
TITLE = UNIV       /SSAD


STUAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  22
PRIME = 0
TABLESIZE(COARSE) =   125 ENTRIES
TABLESIZE(FINE) =   113 ENTRIES
AREASIZE =     10 TABLES PER AREA
LOADFACTOR =     66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =     80 BITS
ENTRYSIZE(FINE) =     88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =    124 SEGMENTS
PACK =
TITLE = UNIV       /SSAD


FACAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  23
PRIME = 0
TABLESIZE(COARSE) =   125 ENTRIES
TABLESIZE(FINE) =   113 ENTRIES
AREASIZE =     10 TABLES PER AREA
LOADFACTOR =     66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =     80 BITS
ENTRYSIZE(FINE) =     88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =    194 SEGMENTS
PACK =
TITLE = UNIV       /SSAD
```

Figure A–3. UNIV Data Base Example (Sheet 13)

```
ADMAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  24
PRIME = 0
TABLESIZE(COARSE) =    125 ENTRIES
TABLESIZE(FINE) =    113 ENTRIES
AREASIZE =     10 TABLES PER AREA
LOADFACTOR =     66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =     80 BITS
ENTRYSIZE(FINE) =     88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =    264 SEGMENTS
PACK =
TITLE = UNIV       /SSAD


BOOKS UNORDERED LIST :
STRUCTURE NUMBER   4
AREASIZE =    500 RECORDS PER AREA
TABLESIZE =      1 ENTRIES
BLOCKSIZE =      5 RECORDS/BLOCK
DATASIZE =    760 BITS
RECORDSIZE =    832 BITS
ENTRYSIZE =    760 BITS
BITS PER BLOCK =  4184


FILE INFORMATION :
FILE NUMBER   1
AREAS =  10
AREASIZE =    300 SEGMENTS
PACK =
TITLE = UNIV       /BOOKS
```

Figure A–3. UNIV Data Base Example (Sheet 14)

COBOL COMPILATION PROCEDURES

The COBOL compilation process requires a program written according to the normal COBOL syntactic conventions, incorporating data set INVOKE statements and the appropriate DMSII statement extensions to COBOL as defined in section 4. During each compilation, upon recognition of an INVOKE statement of a data set, the compiler includes (by copy) into the source program the library file generated by DASDL. The library file contains a complete description of the data set, its sets, automatic subsets, and all of its embedded items, data sets, subsets, and sets. The library files must reside on system disk during compilation. The compiler uses this information to establish the record areas necessary for communication between this program and DMSII. The compilation also provides the information needed for verifying the syntactic correctness of the DMSII statements used in the PROCEDURE DIVISION of the COBOL program.

The COBOL compiler always accesses the library files that have the appropriate identifiers at the time of compilation. If multiple versions of the library files have been produced, it is important that the versions that correspond to the data base versions are the library files loaded at compile time. The versions are checked at execution; if the versions are incompatible, the execution of operations against the data base are disallowed.

The object code produced by the compiler includes a communicate to the MCP for each of the DMSII statements encountered. All DMSII statements are executed by the DMSII access routines within the MCP. Local manipulation of data base items by COBOL statements are handled directly by object code produced by the compiler. The object code produced by the COBOL compiler is ready for execution.

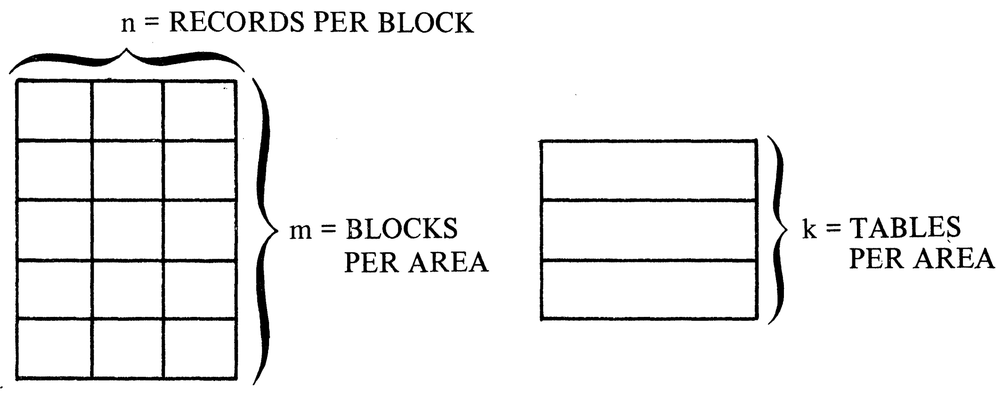
For additional information on COBOL syntax, semantics, options, or compiler operation, refer to the B 1700 Systems COBOL Reference Manual, Form No. 1057197.

DISK FILE STRUCTURES

All disk files used for DMSII data storage are declared as unblocked, 180-byte records. If it is necessary or desirable to print any portion of the data base, an explanation of the file structure is necessary.

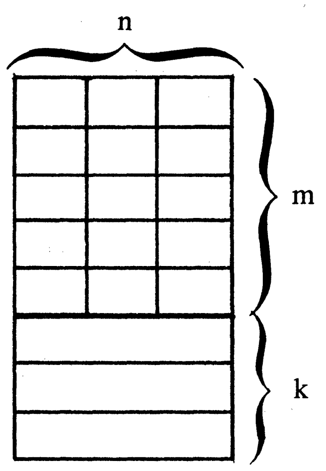
Normal blocking conventions are used. In diagram 1, below, n equals records per block, m equals blocks per area, and k equals tables per area.

Diagram 1:



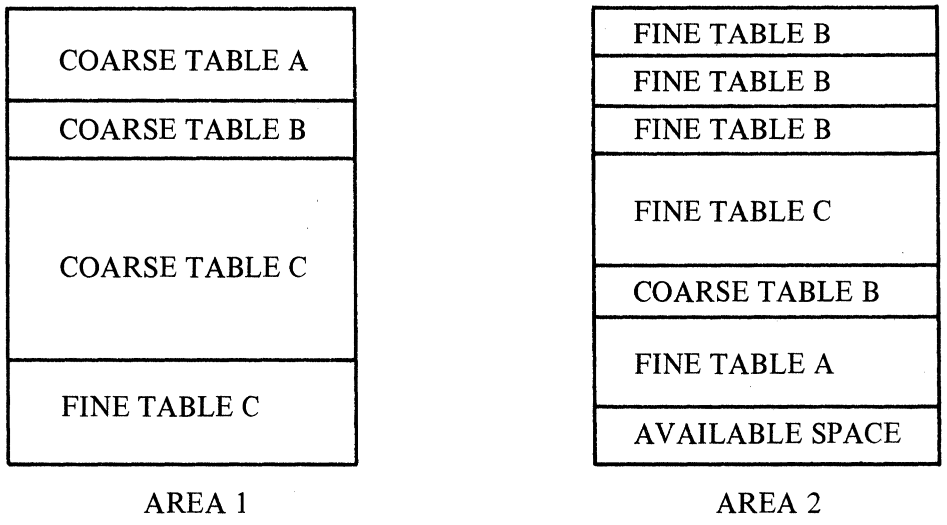
There are two uncommon disk file structure possibilities. The first of these is that the prime index resides in the same file as the data set. Each area is divided into two parts; the first part contains the data set, and the second contains tables as in diagram 2.

Diagram 2:



The second possibility is that multiple index sequential structures reside in the same disk file. All first-level coarse tables are allocated at the front of the file. Additional coarse tables and all fine tables are allocated as required. The results constitute a mixture of coarse and fine tables, as diagram 3 illustrates.

Diagram 3:



DMSII OPERATION AND CONSOLE PRINTER MESSAGES

DMSII is part of the standard MCP. The only special requirement for execution of a DMSII program is that the Data Base Management (DBM) option be set and a Clear/Start performed establishing the DBM environment. This option binds special segments into the Central Service Module (CSM) and is required for DMSII operation. If the option is not present at the time the data base is opened, an appropriate message is displayed.

The data base dictionary must be resident on the system disk at execution. If not, a request for its loading is displayed. Any attempt to open the data base prior to executing the DASDL INITIALIZE statement produces an OPEN operation exception. This exception and most of the other exceptions are returned to the user program for appropriate handling. However, if the user program does not have an On-Exception branch on the DMSII statement producing the exception, the exception results in the termination of the program with an appropriate message displayed.

When the data base disk files are required, DMSII searches the disk directories for the correct files. If the files are not available, a request for the required files is displayed. It is extremely important that the correct version of the data files be loaded. When all processing against a data base has terminated, that is, all programs have closed the data base or gone to EOJ, a message is displayed containing the identifying number of DMSII operations, the number of physical disk reads, the number of physical disk writes, the number of exceptions, and the processor time spent within the DMSII routines.

In the format of DMSII console printer messages, job-specifier is simply used to identify the job for which the message is intended.

A *terminal-reference* indicates that a termination message will be printed. Any time this is indicated, the program must be discontinued; except when the MCP TERM option is set causing the program to terminate automatically. The terminal-reference format is:

⟨*mix-index*⟩: S = ⟨*integer*⟩, D = ⟨*integer*⟩
(@ . . . @, @ . . . @) DS or DP

S indicates the segment number and D indicates the displacement number.

For further information refer to the B 1700 System Software Operational Guide, Form No. 1068731.

DMSII ERROR MESSAGES

DMERROR – DFH STILL IN CHAIN (system malfunction at close-time)

DMS END - - - ⟨*integer*⟩ OPERATIONS ⟨*integer*⟩ READS ⟨*integer*⟩ WRITES and ⟨*integer*⟩ EXCEPTIONS IN ⟨*time*⟩

IO ERROR – DMS WRITE ON STRUCTURE NUMBER ⟨*integer*⟩

⟨*job-specifier*⟩ – BAD FILE NUMBER (incorrect data base DICTIONARY)

⟨*job-specifier*⟩ – PROGRAM DATA BASE VERSION MISMATCH - - - RECOMPILATION OR CORRECT DICTIONARY REQUIRED

⟨*job-specifier*⟩ DMS FILE VERSION MISMATCH ON ⟨*data-file-name*⟩ - - VERSION ON DISK IS *data, time*⟩ - - VERSION REQUIRED IS ⟨*date, time*⟩

⟨*job-specifier*⟩ ⟨*data-base-name*⟩ DATA BASE IS ACTIVE - - - ⟨*data-base-name*⟩ DATA BASE MAY NOT BE OPENED

⟨*job-specifier*⟩ ⟨*data-base-name*⟩ DATA BASE IS NOT INITIALIZED

⟨*job-specifier*⟩ ⟨*data-base-name*⟩ DATA BASE IS NOT AT V.0 LEVEL

⟨*job-specifier*⟩ ⟨*data-base-name*⟩ /DICTIONARY IN USE RECOVERY REQUIRED ON DATA BASE ⟨*data-base-name*⟩

⟨*job-specifier*⟩ – DBM OPTION NOT SET OR CLEAR START REQUIRED

⟨*job-specifier*⟩ – FILE ⟨*data-file-name*⟩ IN USE

⟨*job-specifier*⟩ – INSUFFICIENT DISK SPACE FOR DATA MANAGEMENT

⟨*job-specifier*⟩ – INVALID STRUCTURE NUMBER (incorrect data base DICTIONARY)

⟨*job-specifier*⟩ – NO FILE ⟨*data-base-name*⟩ /DICTIONARY

⟨*job-specifier*⟩ – NO FILE ⟨*data-file-name*⟩

⟨*job-specifier*⟩ – NO PROVISION FOR DMS ⟨DMSTATUS *category*⟩ EXCEPTION – ⟨*terminal-reference*⟩

DATA BASE INTEGRITY

DMSII provides several integrity checks to assist the user in protection of a data base. Each COBOL program accessing a data base is compiled with a specific data base description (referred to as a version) associated with the data base. The version is checked by the system at each execution to ensure the data base dictionary version is the same version as the program. If the versions do not agree, the program is not allowed access to the data base. A version check is made for each structure at the first request to access that structure.

NOTE

> Version checking can be omitted by inserting at DASDL compilation time the $ NO VERSIONCHECK option card. This is usually done while testing the data base, and then only if multiple DASDL executions do not change the data base structure numbers or data sizes. Physical parameters, however, can still be changed when verison checking has been suspended.

The version of a data base file corresponds to the last time the data base was updated. The version information is maintained by DMSII both in the disk file header and the data base dictionary. The disk file header version is updated at the first file update and at the close of an updated file. The data base dictionary version is updated at the close of an updated file. The data base dictionary contains a flag that is set when an update occurs, and is reset when all files having been updated are legitimately closed. This allows DMSII the ability to prevent the use of a data base dictionary if a system failure or a Clear/Start occurs during the updating of any files of the data base. When a system failure is recognized by DMSII, a message requesting the recovery of the data base is displayed on the console printer. This is accomplished by reloading the last version of all updated files, including the data base dictionary. Also, if the version contained in the disk file header does not match the data base dictionary version of the file, the file is not opened and a message is displayed on the console printer requesting the correct version of the file. The DMSII assumes, for checking purposes, that the data base dictionary contains the correct versions. Therefore the data base dictionary file is the central file of the data base and should be saved prior to any update attempt.

LIBRARY MAINTENANCE OF DMSII FILES

In order to protect against the loss of a data base, it is necessary to periodically save a copy of the data base. If system failure occurs during execution of a program that updates the data base or if the disk files are lost or suspected of containing invalid information, the data base must be recovered.

All DMSII files are declared as unblocked, 180-byte records to facilitate the utilization of the standard utilities on these files. SYSTEM/LOADDUMP can be used to save and restore copies of the data base when needed.

Whenever updates to the data base are performed, the data base dictionary file can also be updated. As a consequence it is important to save the dictionary file when any data base files are saved. The dictionary file is the central file in the data base.

At all times the data files must be the same update level as the dictionary file. If, however, a data file has not been updated since it was last saved, it is not necessary to save it again. Care must be used in making this decision as some of the control fields for lists reside in the owner data set.

MEMORY DUMPS

MCPII/ANALYZER has been expanded to list DMSII information whenever memory is dumped while a data base is open. There are two sections of DMSII information in a system dump. The first is the global information maintained by DMSII which includes the following:

a. Pointers to DMS tables and linked lists.

b. Statistics for data base activity.

c. Data base file-identifier.

d. All DMSII disk file headers in use.

e. All in-use structures.

f. Current records and paths for each structure.

g. Lock descriptor table (buffer descriptors).

h. All buffers in memory.

The second section is associated with user programs in the mix. Each program using the data base has a description of a temporary storage area called the DMSII work area. This work area contains the DMSII state information while it executes a data base operation for this program. The record areas associated with each invoked data set are contained within the base and limit register of the program.

Appendix B. DASDL PHYSICAL STRUCTURES

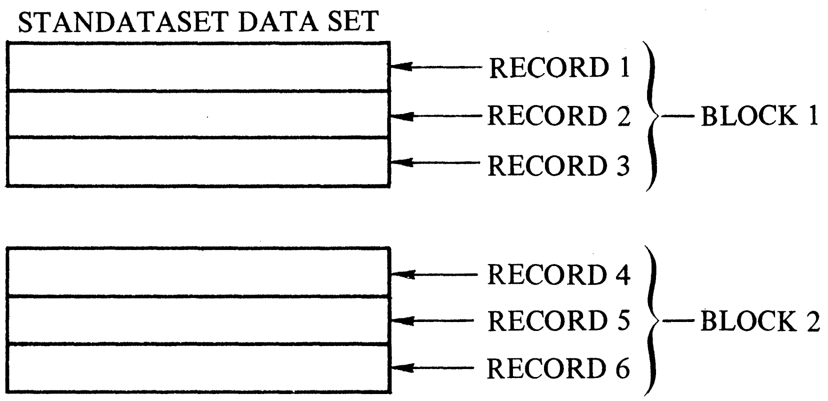
The following DASDL physical structures are examples containing coding syntax, diagrams, and semantics.

DISJOINT DATA SET EXAMPLE

Coding Syntax:

```
STANDATASET DATA SET (
   KEY-FIELD NUMBER (10);
   DATA-FIELD ALPHA (18)
   );
```

Diagram:



Semantics:

a. Records are not ordered.

b. Blocks are not linked together.

c. Available space list maintained in records within the data set.

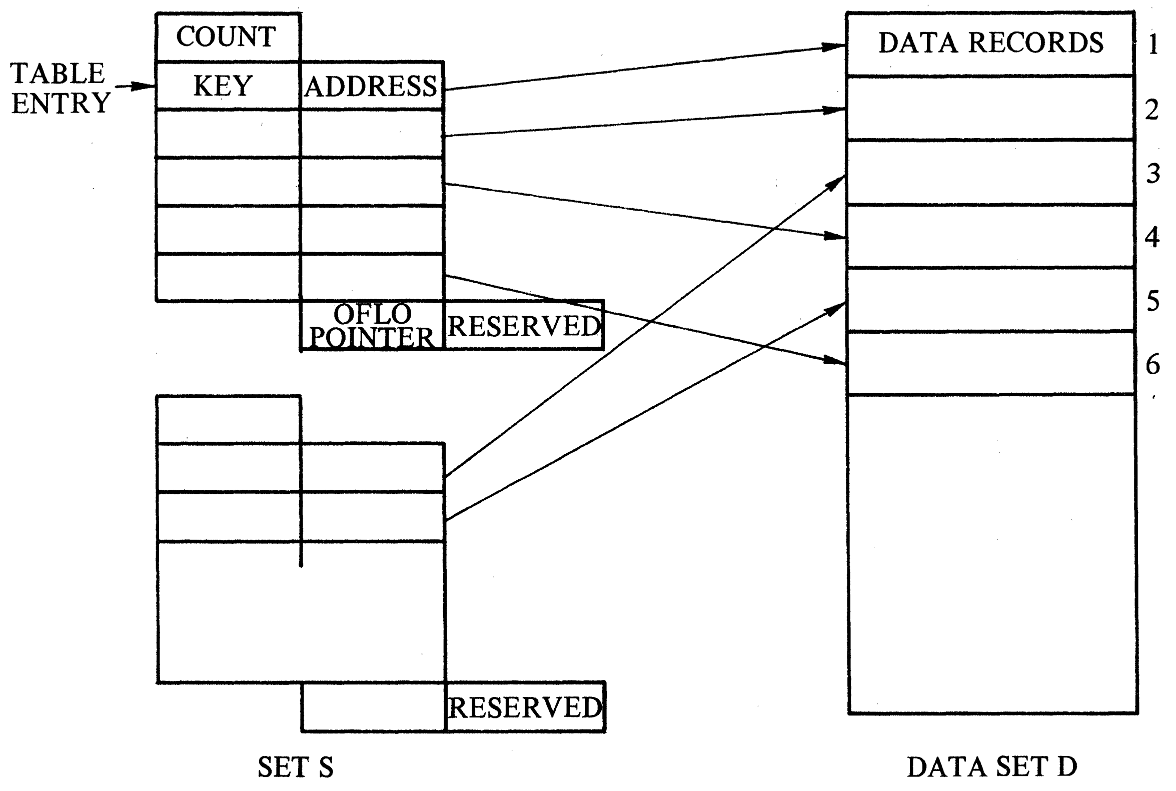
d. In this example, there are three records per block.

INDEX RANDOM EXAMPLE

Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER (10);
  DATA-FIELD (18)
  );
S RETRIEVAL SET OF D KEY (KEY-FIELD);
```

Diagram:



Semantics:

a. MODULUS represents the number of basic tables in the set S.

b. Each table entry contains a symbolic key and an address pointing to the corresponding record in the data set D.

c. Empty entries always reside at the end of the table.

d. LOADFACTOR indicates the degree of sharing of overflow tables that is allowed. When LOADFACTOR equals 50, it indicates that 50 percent of an overflow table is filled by any base tables which overflow, before another overflow table is allocated.

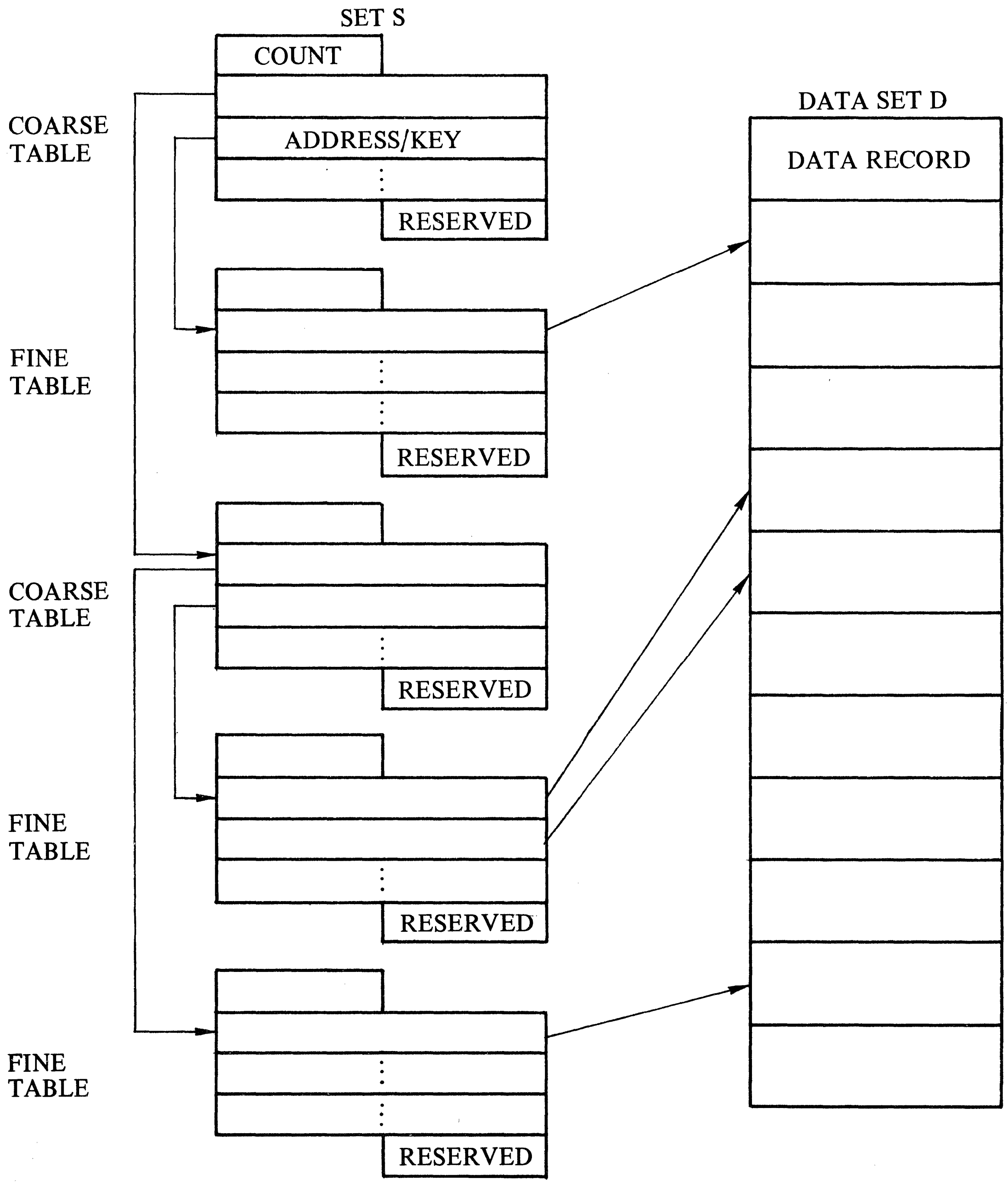
e. COUNT is comprised of 16 bits, OFLO POINTER is comprised of 24 bits, and each ADDRESS is comprised of 32 bits.

INDEX SEQUENTIAL EXAMPLE

Coding Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER (10);
  DATA-FIELD ALPHA (18)
  );
S ORDERED SET OF D KEY (KEY-FIELD);
```

Diagram:



Semantics:

a. Coarse table entries point to a lower level of coarse tables or to fine tables. Fine table entries point to data records in the associated data set.

b. Entries within both tables are in sequence on key value.

c. Table entries consist of addresses and keys.

d. LOADFACTOR specifies the percentage of entries of a fine table which will be filled before another fine table is allocated. For example, if the LOADFACTOR is 50 percent, half of the fine table will be left empty on an initial load for new insertions into the table. Thus, splits may be reduced or eliminated.

e. If a coarse table is full when an attempt is made to add another entry, the coarse table is split into two coarse tables, based on the SPLITFACTOR. For example, given a SPLITFACTOR of 60 percent, 60 percent of the entries of the original coarse table will be moved to the new coarse table, leaving 40 percent in the original coarse table. The normal coarse table split separates duplicate key entries. The split is adjusted so all duplicates remain in the same table. Overflow techniques are not used.

f. COUNT is comprised of 16 bits, addresses for coarse tables are 24 bits, and addresses for fine tables are 32 bits.

g. The key (KEY-FIELD) may be in modified form if it is in descending sequence or a signed number

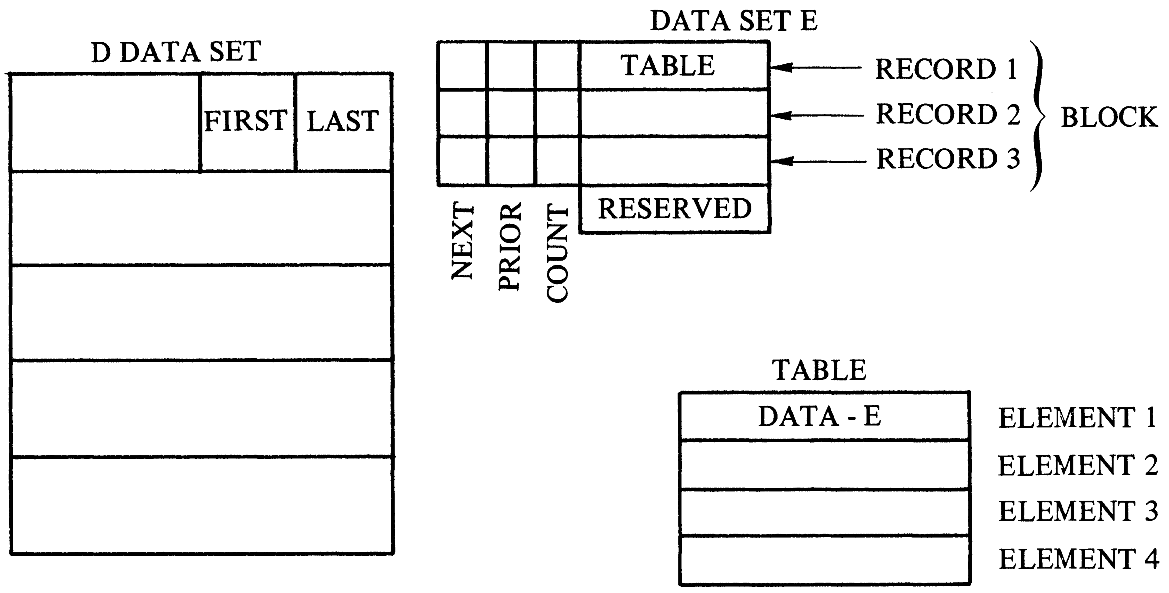
h. AUDIT SERIAL is 32 bits in length and is required.

UNORDERED LIST-EMBEDDED DATA SET EXAMPLE

Coding Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER (10);
  DATA-FIELD ALPHA (18);
  E UNORDERED DATA SET (
    DATA-E ALPHA (18)
  )
);
```

Diagram:



Semantics:

a. The above diagram, which shows an example of an embedded data set as an unordered list, is composed of a collection of tables. Entries within the tables are not ordered.

b. Table entries consist of data records for DATA SET E.

c. Tables are linked together using NEXT and PRIOR fields.

d. All entries within a table belong to the same owner.

e. FIRST and LAST point at tables for an owner.

f. When a table becomes full a new table will be linked in.

g. FIRST, LAST, NEXT, PRIOR are comprised of 32 bits and COUNT is comprised of 8 bits.

h. In this example there are three records per block and four elements per table.

UNORDERED LIST – SUBSET EXAMPLE

Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER  (10);
  DATA-FIELD ALPHA  (18);
  S  SUBSET OF D
);
```

Diagram:

This diagram is the same as the preceding diagram except for the following change to TABLE

TABLE

RECORD ADDRESS

Semantics:

a. The semantics for a subset, as an unordered list, are the same as the semantics for an embedded data set, as an unordered list, except for table entries, which consist of an address only.

ORDERED LIST - EMBEDDED ORDERED DATA SET EXAMPLE

Coding Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER (10);
  DATA-FIELD ALPHA  (18);
  E  ORDERED DATA SET  (
     KEY-E NUMBER  (10);
     DATA-E ALPHA   (18);
  )
  SET-E ACCESS TO E KEY  (KEY-E)
);
```

Diagram:

This diagram is the same as the preceding two diagrams except for the following change to TABLE:

TABLE

DATA - E	KEY - E

Semantics:

a. Entries within the table are ordered by key value.

b. Table entries contain DATA-E. There will be a separate key entry for KEY-E if: (a), the key consists of multiple non-contiguous items; (b), any item is in descending sequence; (c), any item is a signed number. Otherwise, the key item in the record is used.

c. Semantic items c, d, e, f, g, h, and i for an embedded data set as an unordered list also apply for an embedded ordered data set as an ordered list.

d. Full tables can be split in order to maintain ordering.

ORDERED LIST - SUBSET WITH A KEY EXAMPLE

Coding Syntax:

```
D DATA SET (
  KEY-FIELD NUMBER (10);
  DATA-FIELD ALPHA (18);
  S SUBSET OF D KEY (KEY-FIELD)
);
```

Diagram:

This diagram is the same as the preceding three diagrams except for the following change to TABLE:

TABLE

KEY - FIELD	RECORD ADDRESS

Semantics:

a. Entries within the table are ordered by key value.

b. Table entries contain KEY-FIELD and an address.

c. The key (KEY-FIELD) may be in modified form if any items of the key are in descending sequence or a signed number.

d. For a subset with a key, semantic items c, d, e, f, g, h, and i for an embedded data set as an unordered list also apply.

e. Full tables can be split in order to maintain ordering.

Appendix C. DASDL GLOSSARY

The following definitions are intended to give a working description of the terms used in the DASDL section of this manual.

TERM	DEFINITION
ACCESS	A method to reach a desired record of a data set.
DATA SET	A collection of related records. Only data sets have records.
DEADLY EMBRACE	A programmatic condition where two or more programs are simultaneously attempting to lock (prohibit access to) a data record.
DISJOINT	The condition of non-reliance of data sets on the highest level, that is, a data set which is not an item within a data set. Data sets and sets are the only structures that are disjoint. Disjoint sets can only refer to disjoint data sets.
EMBEDDED (INNER LEVEL)	The condition of being dependent on a data set that is on a higher level; that is, a data set which is an item within a data set. An embedded data set can only be referenced by an embedded set on the same level.
INDEX	A table of pointers to a data set used to provide specified access to a data set.
MASTER, PARENT, or OWNER	A data set record which has dependent data sets is referred to as either the master, parent, or owner of the records of the dependent data set. A "master" may itself be a record in an embedded data set. An embedded data set cannot be accessed without accessing the master.
MEMBER	An occurrence of a record of a data set is a member of that data set.
ORDERED	Maintained in a sequence depending on the value of user-specified fields based on a collating sequence.
PATH	An access to a data set record. One instance is a path. A set is an index of paths.
POPULATION	The number of records in a data set. If it is an embedded data set, population is the number of records in the embedded data set per occurrence of its master.
PROPERTIES	The physical structure and parameters of a data set, set, or subset, such as storage requirements or structure type.

TERM	DEFINITION
RECORD	A record contains all the information that pertains to an entity.
SCOPE	The range of influence of a data set, set, or subset.
SET	An index of paths to a data set, with a pointer to each record of that data set.
SPAN	A set points to all records of a data set. A subset need not point to all records of a data set. A subset may only point to some of the records of a data set.
SPLITTING	The method of inserting a new path into a set. The index table is split into two tables rather than through the use of overflow techniques.
SUBSET	A list of paths to records of a data set. The specified records of the data set to be referenced must be programmatically inserted into the subset.
UNORDERED	Not maintained in a user-specified order.

Appendix D. DASDL ERROR MESSAGES AND WARNINGS

INTRODUCTION

The following are lists of error messages and warning messages that can appear on a DASDL output listing.

The bracketed word, ⟨*integer*⟩ , is replaced by the structure number. The structure number is assigned automatically by the compiler to each structure. This number is printed on the DASDL output listing if $STRUCTURE is specified, and it always appears on the COBOL listing.

The bracketed word, ⟨*string*⟩ , is replaced by a reserved word.

NOTE

If the delimiters such as commas, parentheses, or semicolons are misplaced or omitted, the error message can be misleading, and might not indicate the actual error.

WARNING MESSAGES

ATTRIBUTE CHANGED AFTER BEING SET ONCE

FILLER ADDED TO PREVIOUS ITEM

POPULATION OVERFLOWED ON STRUCTURE NUMBER integer 1,000,000 USED INSTEAD

SEQUENCE ERROR

ERROR MESSAGES

ILLEGAL SPECIAL CHARACTER

UNEXPECTED TOKEN IN CONDITIONAL EXPRESSION

INVALID ORDERING KEY FIELD

OPERANDS ARE NOT OF THE SAME TYPE

VERSION MISMATCH, UPDATE NOT DONE

AREASIZE EXCEEDS 2 EXP 16 – 1

BLOCKSIZE EXCEEDS 2 EXP 7 – 1

TABLESIZE EXCEEDS 2 EXP 16 – 1

MAXIMUM SIZE FOR NUMERIC KEY EXCEEDED

DECLARED NUMBER SIZE TOO LARGE

POPULATION EXCEEDS 2 EXP 20 – 1

MORE THAN 16 INDEXES IN ONE FILE

TABLESIZE EXCEEDS 255

COBOL KEY WORD ENCOUNTERED

EOF DDL/DICT – READ

EOF DDL/DICT – WRITE

PARITY ERROR DDL/DICT – WRITE

DATA NAME DICTIONARY OVERFLOW

DUPLICATE FILE NAME-COMPILE ABORTED

DUPLICATE IDENTIFIER FOUND

ONLY ONE VERIFY CLAUSE PER DATA SET

DELIMITERS ")" , ";" OR KEY WORD DUPLICATES REQUIRED HERE

EQUATE SYMBOL EXPECTED NOT FOUND

ITEM TYPE KEY WORD EXPECTED

MISSING KEY NAME

NUMBER EXPECTED NOT FOUND

ATTRIBUTE KEY WORD REQUIRED HERE

SIZE OF IDENTIFIER EXCEEDS 17 CHARACTERS

INCORRECT ATTRIBUTE FOR THIS DECLARATION

NUMBER OF BUFFERS MUST BE GEQ 3 AND LEQ 100

ILLEGAL DECLARATION FOR SUBSET KEY NAME

THIS ITEM NOT ALLOWED IN GROUP ITEM

SUBSET MAY NOT REFERENCE EMBEDDED DATA SETS

NUMBER OF IODESCRIPTORS MUST BE GEQ 2 AND LEQ 20

THIS NAME IS ILLEGAL AS A KEY NAME

REQUIRED DATA ITEM MAY NOT BE SUBSCRIPTED

ILLEGAL $CARD OPTION ENCOUNTERED

SUBSCRIPTED DATA ITEM MAY NOT BE REQUIRED

UNEXPECTED KEY WORD

VERIFY NOT ALLOWED ON SUBSET

RESTART DATA SET MAY NOT BE EMBEDDED

STRUCTURE NUMBER ⟨*integer*⟩ MAY NOT BE IN A DATA SET FILE UNLESS IT IS MADE THE PRIME INDEX

INDEX SEQUENTIAL TABLE SIZE EXCEEDS AREA SIZE FOR FILE

DATA ITEM NAMES AND GROUP ITEM NAMES ARE ILLEGAL HERE

NO OTHER STATEMENTS MAY FOLLOW INITIALIZE STATEMENT

LITERAL IS LARGER THAN OPERAND

DECLARATION FOR KEY NAME IN SUBSET DECLARATION NOT FOUND

DATA SET REFERENCED BY SUBSET NOT FOUND

MISSING ORDERING KEY

ACCESS PATH CANNOT BE CHANGED

MISSING COMMA

COMMENT NOT ALLOWED IN THIS CONTEXT

DATA SET NAME NEEDED HERE

IDENTIFIER EXPECTED NOT FOUND

IDENTIFIER NOT FOUND OR UNDEFINED

EMBEDDED DATA SETS AND SUBSETS NOT ALLOW IN RESTART DATA SET

TYPE ON EMBEDDED DATA SET MISSING ORDERED OR UNORDERED REQUIRED HERE

KEY TYPE MAY BE MISSING OR MISSPELLED

LISTS AND DATA SETS MAY NOT BE MIXED

MISSING LEFT PARENTHESIS

ORDERING KEY NAME NEEDED HERE

MISSING OPERATOR IN CONDITIONAL CLAUSE

KEY WORD ORDERED OR UNORDERED REQUIRED

RETRIEVAL KEY NAME NEEDED HERE

MISSING RIGHT PARENTHESIS

KEY WORD ALL REQUIRED HERE

KEY WORD DATA REQUIRED HERE OR KEY WORDS ORDERED OR RETRIEVAL REQUIRED

KEY WORD FOR REQUIRED HERE

MISSING KEY WORD KEY

KEY WORD OF REQUIRED HERE

KEY WORD ORDERED REQUIRED HERE

KEY WORD SET REQUIRED HERE

KEY WORD TIMES REQUIRED HERE

KEY WORD TO REQUIRED HERE

KEY WORD TRAIL REQUIRED HERE

KEY WORD UNORDERED REQUIRED HERE

RESERVED WORD WHERE REQUIRED HERE

MISSING SEMICOLON

SLASH REQUIRED BETWEEN FILE NAMES

AREASIZE FOR STRUCTURE NUMBER ⟨integer⟩ EXCEEDS 65,535

SUBSCRIPTED DATA ITEMS MAY NOT BE USED AS KEYS

ILLEGAL TOKEN IN OPTION STATEMENT

ILLEGAL ITEM IN PARAMETER LIST

NUMBER TOO LARGE – MAX 8 CHARACTERS

VALUE IN OCCURS EXCEEDS 1023

ATTEMPTED TO ASSIGN NEW VALUE TO POPULATION

ONLY ONE DATA SET IS ALLOWED IN A FILE

ONLY ONE KEY ALLOWED IN AN EMBEDDED SET

ONLY ONE INDEX IS ALLOWED WITH A DATA SET

THIS INDEX MUST BE IN A FILE BY ITSELF

THIS LIST MUST BE IN A FILE BY ITSELF

ORDERED DATA SET MUST HAVE AN ACCESS PATH

ORDERING TYPE AND/OR ITEM NAME LIST MISSING

" REQUIRED ON CONTINUATION CARD

AREASIZE MAY NOT BE LESS THAN BLOCKSIZE

AUDIT FILE AREASIZE MUST BE 2 OR MORE

A STRUCTURE IDENTIFIER MAY ONLY APPEAR IN ONE FILE STRUCTURE LIST

DESCRIPTION TOO LONG – MAX 172 CHAR

NUMBER OF AREAS EXCEEDS 105

BITS PER BLOCK EXCEEDS 65,535 IN STRUCTURE NUMBER ⟨*integer*⟩

UNDEFINED IDENTIFIER

UNEXPECTED PARENTHESIS ENCOUNTERED

UNEXPECTED KEY WORD TOKEN SYMBOL ENCOUNTERED

UNEXPECTED SEMICOLON ENCOUNTERED

SET MUST APPEAR IMMEDIATELY AFTER DATA SET DESCRIPTION

UNORDERED EMBEDDED DATA SET MAY NOT HAVE AN ACCESS PATH

Appendix E. COBOL EXTENSIONS

The COBOL Procedure Division has been extended to provide an interface between a COBOL program and a data base. A brief synopsis of these extensions is given in table E-1.

Table E-1. COBOL Extensions

Extension	Definition
OPEN	Used to open a data base for subsequent access and to specify access mode. A data base can not be accessed until it has been opened.
FIND	Used to read a record from a data set.
MODIFY	Same as FIND, except the record is locked against concurrent modification by another user.
STORE	Used to write a new record into a data set or to replace an existing record in a data set with a modified copy of that record.
DELETE	Used to remove a record from a data set.
FREE	Used to unlock a record.
CLOSE	Used to close a data base when further access is no longer required.
CREATE	Used to initialize the record area of a data set.
RECREATE	Used to initialize the data set, set, and subset items of the record area of a data set. All data items remain unaltered.
INSERT	Used to insert a record into a manual subset.
REMOVE	Used to remove a record from a manual subset.

Appendix F. QUALIFYING A DATA BASE DESCRIPTION

Unique identifiers are required in COBOL programs. If a data set is invoked more than once, different internal names must be used in order that items within the data set can be appropriately qualified.

A variable declaration with the same name as a data base item can be used only if the item is able to be uniquely qualified.

In a selection expression, sets and subsets require qualification if they are not unique identifiers. Data base items in a selection expression need not be qualified.

Example:

DASDL

```
D1  DATA SET (
    A NUMBER  (5);
    B NUMBER (3));
S1  ORDERING SET OF D1 KEY (A);
```

COBOL

```
DB  DBASE.
01  D1 INVOKE D1.
01  DA INVOKE D1.
WORKING-STORAGE SECTION.
77 A PIC 99. (Invalid because it can not be uniquely qualified.)
01 Q.
   03 A PIC 99. (Valid because it can be qualified.)
```

PROCEDURE DIVISION.

```
MOVE A OF D1 TO L. (Valid.)
FIND S1 OF D1 AT A = L. (Valid.)
MOVE A TO L. (Insufficient qualification of A.)
FIND S1 AT A = L. (Insufficient qualification of S1.)
FIND S1 OF DA AT A OF DA = L. (Valid but A need not be qualified in a selection expression.)
```

Appendix G. B 1700 – B 6700/B 7700 DMSII COMPATIBILITY

The relationship of B 1700 DMSII to B 6700/B 7700 DMSII is as follows:

a. B 1700 DMSII is a logical subset of B 6700/B 7700 DMSII.

b. Any COBOL constructs used to access B 1700 DMSII are syntactically and semantically compatible with B 6700/B 7700 DMSII.

c. Any physical data bases developed on the B 1700 DMSII are not compatible with B 6700/B 7700 formats.

d. The ordered embedded data set, together with its access set of B 1700 DMSII, is not supported by B 6700/B 7700 DMSII. However, the identical COBOL capability is provided by making an ordered embedded data set an unordered embedded data set together with a set on B 6700/B 7700 DMSII.

e. The physical mapping algorithms on the two systems differ significantly and the physical mapping parameters should be reviewed carefully prior to transfer from B 1700 DMSII to B 6700/B 7700 DMSII. For example, B 1700 DMSII SPLIT FACTOR, STORAGE, and PRIME generate warning messages, and are then ignored on the B 6700/B 7700 series systems.

f. Ordered and retrieval set types are not meaningful on B 6700/B 7700 DMSII. They produce a regular B 6700/B 7700 DMSII set.

g. DASDL Parameters differ significantly, and there is no direct correspondence between B 1700 DMSII and B 6700/B 7700 DMSII.

Data bases should be remapped and must be reloaded at the time of transfer to B 6700/B 7700 DMSII. However, any DMSII statements in COBOL programs developed for B 1700 DMSII are valid on B 6700/ B 7700 DMSII.

Appendix H. DATA BASE EXAMPLE

INTRODUCTION

The development of this example follows the same process used in the development of any data base. The first step is to become thoroughly acquainted with the problem, the input and data available, and the output or information required. What other information might be desired? What are the processing requirements? Are there any time critical or volume critical demands? What are the primary requests? What are the entities and their relationships? Identify the major properties of each entity.

The data base example is concerned with the business administration department of a small university. The major elements of this example are students (MSF), courses (UNIV-COURSES), and personnel (UNIV-PERSONNEL). The major component of MSF is achievement by quarter (QUARTER). The achievement consists of the courses (CORSES) taken in a given quarter (QUARTER). The students (STUDENTS) attending, the books (BOOKS) required, and the professor (PROFESSOR) teaching are the major attributes of university courses (UNIV-COURSES). University personnel (UNIV-PERSONNEL) are of interest due to the courses (CORSES) taught. The supervisor (SUPR) is also of interest.

The next step is to draw a diagram of the data base similar to the one illustrated in Figure H-1. All major entities are shown as boxes, MSF, UNIV-PERSONNEL, and UNIV-COURSES. It is then possible to determine for each major property of these items whether it is a new entity or a relationship between existing entities. All new entities become boxes (QUARTER, CORSES, and BOOKS). CORSES is a particular student achievement in a particular class in a particular quarter. Thus, it is an entity, not a relationship to UNIV-COURSES.

These entities which have just been added to the data base are connected to their major entities by means of a broken line. This indicates that they have meaning only when taken in context of a major entity. For example, it is only meaningful to examine BOOKS in context of a particular UNIV-COURSES.

The output of DASDL code is the final step. Remaining, is the optimization of the data base. There are two parts to this: one, optimizing the logical structures, and two, optimizing the physical mapping. The addition of ADR is an example of logical optimization. This type of optimization may continue through testing of the data base. However, these changes may require changes to any programs which have been developed to interface to that part of the data base.

Modifications to the physical mapping should not begin until after the default mapping has been reviewed. Testing and performance measurement also indicates where optimization is required. However, a permanent data base does not exist until all modifications have been made. COBOL programs do not require changes for physical optimization, although recompilation can be required.

Figure H-2 contains a DASDL listing of the UNIV data base; Figure H-3 is a COBOL program intended to access the UNIV data base. Also included is an explanation of the identifiers used in the UNIV data base example.

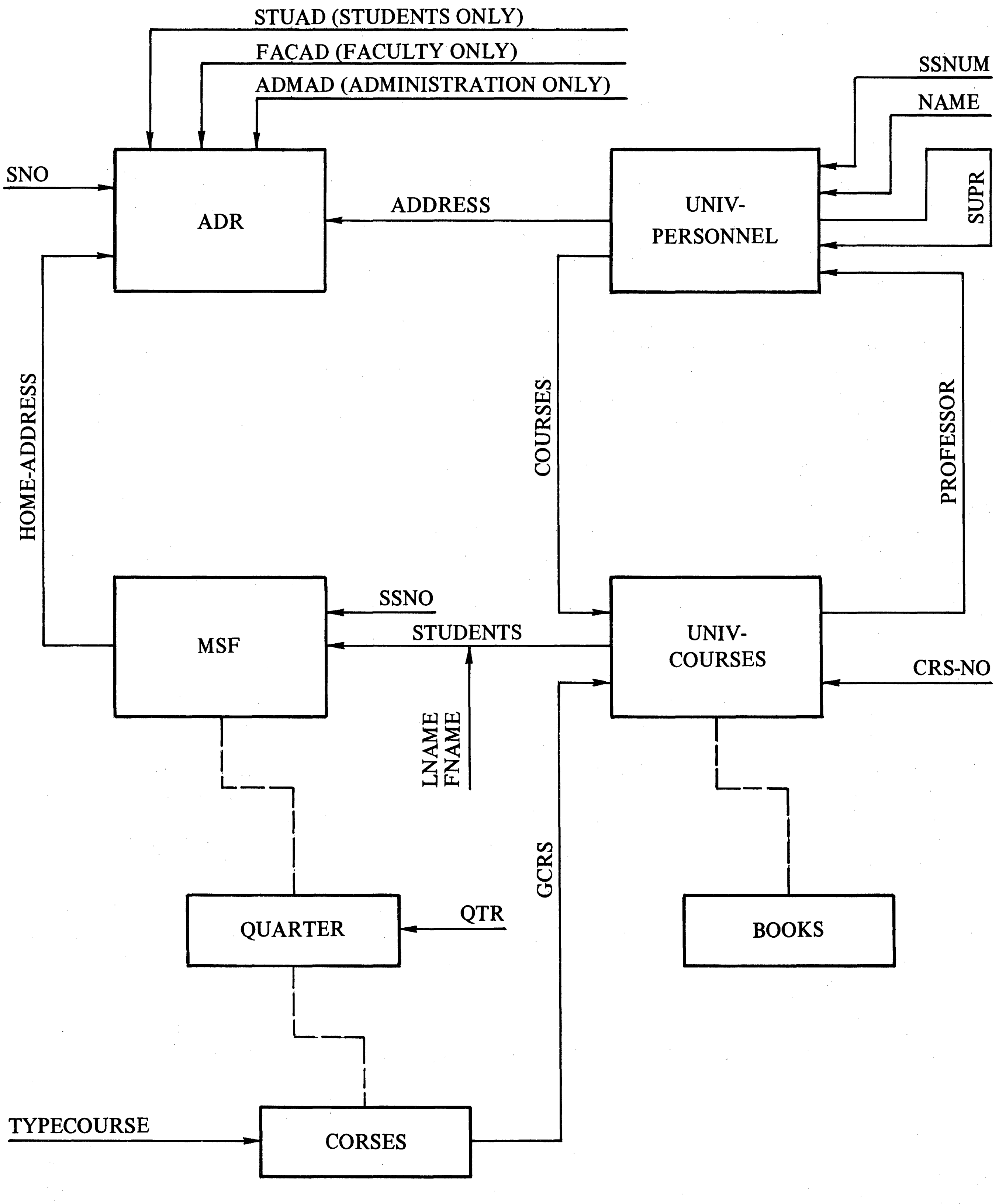


Figure H-1. UNIV Data Base Diagram

All the relationships are represented by solid arrows connecting two entities. Thus, STUDENTS, PROFESSOR, COURSES, and SUPR are added to the data base. GCRS (a pointer to UNIV-COURSES) is also added for access to common information about the UNIV-COURSES from a particular CORSES record.

The address of a student (MSF) or UNIV-PERSONNEL would usually be thought of as a group item with the record of MSF or UNIV-PERSONNEL. However, by reviewing the volume of information and the infrequent use of the information, the data base may be optimized. All addresses (ADR) are stored in a separate data set, with pointers from MSF and UNIV-PERSONNEL to the appropriate ADR.

The last addition to the diagram is to indicate for each entity and relationship the keys which are of interest. These keys allow access to the records in order of key value. The keys of interest are added to the diagram by means of short arrows. The new additions are SNO of ADR, SSNUM and NAME of UNIV-PERSONNEL, CRS-NO of UNIV-COURSES, SSNO of MSF, LNAME and FNAME of STUDENTS, QTR of QUARTER, and TYPECOURSE of CORSES. Of special note is the requirement to access addresses by type of person. This leads to the definition of three different access paths to ADR: one for students only, one for faculty, and one for administration. These paths are referenced as STUAD, FACUD, and ADMAD of ADR.

When the data base diagram is completed, the next task is to translate it into DASDL. The mapping is as follows:

a. All boxes (entities) are data sets; those that are connected by broken lines become embedded data sets. Thus, CORSES is an embedded data set of QUARTER which in turn is an embedded data set of MSF.

b. All arrows (relationships) between two entities become manual subsets. For example, PROFESSOR is a manual subset of UNIV-PERSONNEL.

c. All short arrows on disjoint data sets having conditions for inclusion become automatic subsets. For example, STUAD is the address of only the students.

d. All other arrows on disjoint data sets are subdivided into those on which access of the records in order of the key value is required, and those which have no ordering requirements. All of the former map into ordered sets; the latter become retrieval sets. For example, MSFSET is an ordered set providing access to MSF in SSNO sequence. An ordered set also allows access by key value. Thus, for example, it is valid to retrieve a MSF record where SSNO = 123456789.

e. All arrows on embedded data sets are mapped as access sets. For example, TYPECOURSE is mapped by CSET of CORSES.

f. All arrows on subsets add a key structure to that manual subset.

g. A further requirement added to the DASDL input is that all courses must have a positive number of credits assigned to them, have a class size less than or equal to 60, and have some professors assigned to teach the class. This is expressed by the VERIFY clause. The requirement that TYPECOURSE and GRADE be known for all students (MSF) CORSES is indicated by the REQUIRED option. This is an indication of the validity-checking options available for usage in the example.

```
 UNIV       DATA BASE
$LIST SINGLE

$SUPPRESS
$FILE STRUCTURE
%THIS DASDL PROGRAM GIVES EXAMPLES
%OF THE VARIOUS CONSTRUCTS USED IN
%DASDL TO DESCRIBE A DATA BASE
PARAMETERS(
    BUFFERS = 10   );
UNIV-COURSES DATA SET "MAIN FILE" (
    CRS-NAME GROUP (
         DEPARTMENT ALPHA(2);
         LEVEL NUMBER(3);
         CRS-NO NUMBER(4));
    NOPROF NUMBER(2);
    DAYS-OF-WEEK GROUP (
         MON NUMBER(1);
         TUES NUMBER(1);
         WEDS NUMBER(1);
         THURS NUMBER(1);
         FRI NUMBER(1);
         SAT NUMBER(1));
    BUILDING NUMBER(3);
    ROOMNUMBER ALPHA(2);
    COURSENAME ALPHA(24);
    FLAG-BITS ALPHA(12);
    HOURSCRDT NUMBER(4);
    CLASS-SIZE NUMBER(2);
    PROFESSOR SUBSET OF UNIV-PERSONNEL,POPULATION = 3;
    BOOKS UNORDERED DATA SET(
         LC NUMBER(9);
         TITLES ALPHA(60);
         AUTHR ALPHA(30));
    STUDENTS SUBSET OF MSF KEY IS
         (LNAME,FNAME)DUPLICATES,
         POPULATION = 300)
    POPULATION = 1000
VERIFY((HOURSCRDT GTR 0 AND CLASS-SIZE LEQ 60) AND NOPROF NEQ 0);
    UNIV-C-SET ORDERED SET OF UNIV-COURSES KEY IS (CRS-NO);
UNIV-PERSONNEL DATA SET(
    NAME GROUP(
         LASTNAME ALPHA(15);
         FIRSTNAME ALPHA(10));
    SEX NUMBER(1);
    AGE NUMBER(2);
    SSNUM NUMBER(9);
    DPT ALPHA(4);
    RANK ALPHA(1);
    SALARY NUMBER(S7,2);
    COURSES SUBSET OF UNIV-COURSES,POPULATION = 8;
    ADDRES SUBSET OF ADR;
    SUPR SUBSET OF UNIV-PERSONNEL);
    SS-U-P ORDERED SET OF UNIV-PERSONNEL KEY IS (SSNUM);
    U-P-SET ORDERED SET OF UNIV-PERSONNEL KEY IS
         (LASTNAME,FIRSTNAME) DUPLICATES;
MSF DATA SET(
    SSNO NUMBER(9);
    NONAM NUMBER(1);
    LNAME ALPHA(30);
    MNAME ALPHA(30);
```

Figure H–2. DASDL Program Example (Sheet 1)

```
    FNAME ALPHA(30);
    CAMPUS-ADDRESS GROUP(
         DORM ALPHA(6);
         ROOM NUMBER(4);
         POBOX NUMBER(4);
         PHONE NUMBER(7));
    ND NUMBER(2);
    DEGREE ALPHA(4) OCCURS 6 TIMES;
    TOTHRS NUMBER(3);
    TOTGP NUMBER(3);
    GRADE-POINT-AVG NUMBER(3,2);
    MJR NUMBER(3);
    AMJR ALPHA(18);
    SSEX NUMBER(1);
    SAGE NUMBER(2);
    HOME-ADDRESS SUBSET OF ADR;
    QUARTER   ORDERED DATA SET(
         QTR ALPHA(4);
         QTTRHRS NUMBER(2);
         QTRGP NUMBER(2);
         CORSES ORDERED DATA SET(
              TYPECOURSE NUMBER(1) REQUIRED;
              GRADE ALPHA(2) REQUIRED;
              CRS SUBSET OF UNIV-COURSES;
              YR NUMBER(2);
              Q NUMBER(2);
              GCRS SUBSET OF UNIV-COURSES;
              GGD ALPHA(2);
              TITLE-OF-PAPER ALPHA(30);
              PPRGD ALPHA(2));
              POPULATION = 4;
              CSET ACCESS TO CORSES KEY IS
                   (TYPECOURSE) DUPLICATES);
         POPULATION = 5000;
         QSET ACCESS TO QUARTER KEY IS (QTR));
    MSFSET ORDERED SET OF MSF KEY IS (SSNO);
ADR DATA SET(
    FACULTY-STUDENT NUMBER(1);
    SNO NUMBER(9) REQUIRED;
    ADLN ALPHA(54) OCCURS 9 TIMES;
    ZIPC NUMBER(5) REQUIRED;
    PHON NUMBER(10));
SAD ORDERED SET OF ADR KEY IS (ZIPC) DUPLICATES;
    SSAD ORDERED SET OF ADR KEY IS (SNO);
STUAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 1) KEY IS (ZIPC,SNO)
                                                        DUPLICATES;
FACAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 2) KEY IS (ZIPC,SNO)
                                                        DUPLICATES;
ADMAD SUBSET OF ADR WHERE(FACULTY-STUDENT EQL 3) KEY IS (ZIPC,SNO)
                                                        DUPLICATES;
BOOKS(
    AREASIZE = 500,
    TYPE = UNORDERED LIST,
    BLOCKSIZE = 5);
BOOKFILE STORAGE FOR BOOKS(
    TITLE = UNIV/LIBRARY,
    AREAS = 10);
UNIV-C-SET(
    TABLESIZE = 12,
    AREASIZE = 10,
    TYPE = INDEX SEQUENTIAL,
```

Figure H–2. DASDL Program Example (Sheet 2)

```
00011300 ;    LOADFACTOR = 9);
00011400 ;UNIV-PERSONNEL(
00011450 ;    PRIME,
00011500 ;    POPULATION = 997);
00011600 ;INITIALIZE;
```

```
                                UNIV-COURSES DATA SET :
                                STRUCTURE NUMBER   1
                                PRIME = 0
                                BLOCKSIZE =      2 RECORDS/BLOCK
                                AREASIZE =   292 RECORDS PER AREA
                                POPULATION =     1000 RECORDS
                                DATASIZE =   424 BITS
                                RECORDSIZE =   616 BITS
                                BITS PER BLOCK =  1232


                                UNIV-C-SET INDEX SEQUENTIAL SET :
                                STRUCTURE NUMBER   7
                                PRIME = 1
                                TABLESIZE(COARSE) =     14 ENTRIES
                                TABLESIZE(FINE) =     12 ENTRIES
                                AREASIZE =    10 TABLES PER AREA
                                LOADFACTOR =      8 PERCENT
                                SPLITFACTOR = 50 PERCENT
                                ENTRYSIZE(COARSE) =    40 BITS
                                ENTRYSIZE(FINE) =    48 BITS
                                BITS PER BLOCK(COARSE) =   576
                                BITS PER BLOCK(FINE) =   592


                                FILE INFORMATION :
                                FILE NUMBER   2
                                AREAS =  20
                                AREASIZE =   156 SEGMENTS
                                PACK =
                                TITLE = UNIV      /UNIV-COURS


                                PROFESSOR UNORDERED LIST :
                                STRUCTURE NUMBER   2
                                AREASIZE =    50 RECORDS PER AREA
                                TABLESIZE =    42 ENTRIES
                                BLOCKSIZE =     8 RECORDS/BLOCK
                                DATASIZE =    32 BITS
                                RECORDSIZE =  1416 BITS
                                ENTRYSIZE =    32 BITS
                                BITS PER BLOCK = 11360


                                FILE INFORMATION :
                                FILE NUMBER   3
                                AREAS =  20
                                AREASIZE =    50 SEGMENTS
                                PACK =
                                TITLE = UNIV      /PROFESSOR


                                STUDENTS ORDERED LIST :
                                STRUCTURE NUMBER   5
```

Figure H–2. DASDL Program Example (Sheet 3)

```
AREASIZE =  7500 RECORDS PER AREA
TABLESIZE =     2 ENTRIES
BLOCKSIZE =     1 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1096 BITS
ENTRYSIZE =   512 BITS
BITS PER BLOCK =  1128


FILE INFORMATION :
FILE NUMBER   4
AREAS =  20
AREASIZE =  7500 SEGMENTS
PACK =
TITLE = UNIV      /STUDENTS


UNIV-PERSONNEL DATA SET :
STRUCTURE NUMBER   3
PRIME = 1
BLOCKSIZE =     2 RECORDS/BLOCK
AREASIZE =  1192 RECORDS PER AREA
POPULATION =      997 RECORDS
DATASIZE =   320 BITS
RECORDSIZE =   512 BITS
BITS PER BLOCK =  1024


FILE INFORMATION :
FILE NUMBER   5
AREAS =  20
AREASIZE =   596 SEGMENTS
PACK =
TITLE = UNIV      /UNIV-PERSO


COURSES UNORDERED LIST :
STRUCTURE NUMBER   8
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   6
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /COURSES


ADDRES UNORDERED LIST :
STRUCTURE NUMBER   9
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
```

Figure H–2. DASDL Program Example (Sheet 4)

```
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   7
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV       /ADDRES


SUPR UNORDERED LIST :
STRUCTURE NUMBER  11
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER   8
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV       /SUPR


MSF DATA SET :
STRUCTURE NUMBER   6
PRIME = 0
BLOCKSIZE =     1 RECORDS/BLOCK
AREASIZE =  1191 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  1276 BITS
RECORDSIZE =  1404 BITS
BITS PER BLOCK =  1404


MSFSET INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  19
PRIME = 1
TABLESIZE(COARSE) =    143 ENTRIES
TABLESIZE(FINE) =    119 ENTRIES
AREASIZE =    12 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    60 BITS
ENTRYSIZE(FINE) =    72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


FILE INFORMATION :
FILE NUMBER   9
AREAS =  20
AREASIZE =  1263 SEGMENTS
```

Figure H–2. DASDL Program Example (Sheet 5)

```
PACK =
TITLE = UNIV      /MSF


HOME-ADDRESS UNORDERED LIST :
STRUCTURE NUMBER  14
AREASIZE =   125 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER  10
AREAS =  20
AREASIZE =   125 SEGMENTS
PACK =
TITLE = UNIV      /HOME-ADDRE


CRS UNORDERED LIST :
STRUCTURE NUMBER  17
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER  11
AREAS =  20
AREASIZE =  9632 SEGMENTS
PACK =
TITLE = UNIV      /CRS


GCRS UNORDERED LIST :
STRUCTURE NUMBER  18
AREASIZE =  9632 RECORDS PER AREA
TABLESIZE =    42 ENTRIES
BLOCKSIZE =     4 RECORDS/BLOCK
DATASIZE =    32 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =    32 BITS
BITS PER BLOCK =  5696


FILE INFORMATION :
FILE NUMBER  12
AREAS =  20
AREASIZE =  9632 SEGMENTS
PACK =
TITLE = UNIV      /GCRS
```

Figure H–2. DASDL Program Example (Sheet 6)

```
CORSES ORDERED LIST :
STRUCTURE NUMBER  16
AREASIZE = 14339 RECORDS PER AREA
TABLESIZE =     3 ENTRIES
BLOCKSIZE =     1 RECORDS/BLOCK
DATASIZE =   312 BITS
RECORDSIZE =  1392 BITS
ENTRYSIZE =   440 BITS
BITS PER BLOCK =  1424


FILE INFORMATION :
FILE NUMBER  13
AREAS =  20
AREASIZE = 14339 SEGMENTS
PACK =
TITLE = UNIV      /CORSES


QUARTER ORDERED LIST :
STRUCTURE NUMBER  15
AREASIZE = 20850 RECORDS PER AREA
TABLESIZE =    12 ENTRIES
BLOCKSIZE =     1 RECORDS/BLOCK
DATASIZE =    48 BITS
RECORDSIZE =  1416 BITS
ENTRYSIZE =   112 BITS
BITS PER BLOCK =  1448


FILE INFORMATION :
FILE NUMBER  14
AREAS =  20
AREASIZE = 41700 SEGMENTS
PACK =
TITLE = UNIV      /QUARTER


ADR DATA SET :
STRUCTURE NUMBER  10
PRIME = 0
BLOCKSIZE =     1 RECORDS/BLOCK
AREASIZE =   910 RECORDS PER AREA
POPULATION =   10000 RECORDS
DATASIZE =  3988 BITS
RECORDSIZE =  3988 BITS
BITS PER BLOCK =  3988


SAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  20
PRIME = 1
TABLESIZE(COARSE) =   129 ENTRIES
TABLESIZE(FINE) =   101 ENTRIES
AREASIZE =    11 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    44 BITS
ENTRYSIZE(FINE) =    56 BITS
BITS PER BLOCK(COARSE) =  5692
BITS PER BLOCK(FINE) =  5672
```

Figure H–2. DASDL Program Example (Sheet 7)

```
FILE NUMBER  15
AREAS =  20
AREASIZE =  2774 SEGMENTS
PACK =
TITLE = UNIV      /ADR


SSAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  21
PRIME = 0
TABLESIZE(COARSE) =   143 ENTRIES
TABLESIZE(FINE) =   119 ENTRIES
AREASIZE =     9 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    60 BITS
ENTRYSIZE(FINE) =    72 BITS
BITS PER BLOCK(COARSE) =  8596
BITS PER BLOCK(FINE) =  8584


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =    54 SEGMENTS
PACK =
TITLE = UNIV      /SSAD


STUAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  22
PRIME = 0
TABLESIZE(COARSE) =   125 ENTRIES
TABLESIZE(FINE) =   113 ENTRIES
AREASIZE =    10 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    80 BITS
ENTRYSIZE(FINE) =    88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =  124 SEGMENTS
PACK =
TITLE = UNIV      /SSAD


FACAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  23
PRIME = 0
TABLESIZE(COARSE) =   125 ENTRIES
TABLESIZE(FINE) =   113 ENTRIES
AREASIZE =    10 TABLES PER AREA
LOADFACTOR =    66 PERCENT
```

Figure H–2. DASDL Program Example (Sheet 8)

```
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    80 BITS
ENTRYSIZE(FINE) =    88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =   194 SEGMENTS
PACK =
TITLE = UNIV      /SSAD


ADMAD INDEX SEQUENTIAL SET :
STRUCTURE NUMBER  24
PRIME = 0
TABLESIZE(COARSE) =   125 ENTRIES
TABLESIZE(FINE) =   113 ENTRIES
AREASIZE =    10 TABLES PER AREA
LOADFACTOR =    66 PERCENT
SPLITFACTOR = 49 PERCENT
ENTRYSIZE(COARSE) =    80 BITS
ENTRYSIZE(FINE) =    88 BITS
BITS PER BLOCK(COARSE) = 10016
BITS PER BLOCK(FINE) =  9960


FILE INFORMATION :
FILE NUMBER  16
AREAS =  20
AREASIZE =   264 SEGMENTS
PACK =
TITLE = UNIV      /SSAD


BOOKS UNORDERED LIST :
STRUCTURE NUMBER   4
AREASIZE =   500 RECORDS PER AREA
TABLESIZE =     1 ENTRIES
BLOCKSIZE =     5 RECORDS/BLOCK
DATASIZE =   760 BITS
RECORDSIZE =   832 BITS
ENTRYSIZE =   760 BITS
BITS PER BLOCK =  4184


FILE INFORMATION :
FILE NUMBER   1
AREAS =  10
AREASIZE =   300 SEGMENTS
PACK =
TITLE = UNIV      /BOOKS
```

Figure H–2. DASDL Program Example (Sheet 9)

```
001001   IDENTIFICATION DIVISION.
001002   PROGRAM-ID. DMSCOBOLSAMPLE.
001003   ENVIRONMENT DIVISION.
001004   INPUT-OUTPUT SECTION.
001005   FILE-CONTROL.
001006       SELECT CARD ASSIGN TO READER.
001007       SELECT MONITOR-DMS ASSIGN TO PRINTER.
001008   DATA DIVISION.
001009   FILE SECTION.
001010   FD  CARD.
001011   01  CARD-REC.                                                                  000,0000066
001012       03  C-TYPE          PIC 9.                                          [0001] 000,0000066
001013       03  C-SSNO          PIC 9(9).                                       [0002] 000,0000068
001014       03  C-GRD-PT-AVG    PIC 999V99.                                     [0003] 000,0000086
001015       03  C-SEX           PIC X.                                                 000,0000096
001016       03  C-AGE           PIC 99.                                                000,0000098
001017       03  C-QTR           PIC X(4).                                       [0004] 000,0000102
001018       03  C-TYPECOURSE    PIC 9.                                                 000,0000110
001019       03  C-GRADE         PIC XX.                                                000,0000112
001020       03  C-TITLE-PAPER   PIC X(30).                                             000,0000116
001021       03  C-NAME          PIC X(24).                                      [0005] 000,0000176
001022   FD  MONITOR-DMS.
001023   01  MONITOR-REC.                                                        [0006] 000,0000224
001024       03  MONITOR-EXCEPTION   PIC X(4).                                          000,0000224
001025       03  MONITOR-STATUS-B    PIC X(20).                                  [0007] 000,0000232
001026       03  MONITOR-STATUS      PIC ZZZ9BB.                                        000,0000272
001027       03  MONITOR-VERB        PIC X(20).                                  [0008] 000,0000284
001028       03  MONITOR-SET         PIC X(17).                                         000,0000324
001029       03  MONITOR-STRUCTURE   PIC 9(3).                                          000,0000358
001030       03  FILLER              PIC X(64).                                         000,0000364
001031   DATA-BASE SECTION.
001032   DB  UNIV.
001033   01 MASTER INVOKE MSF.

*        01 MSF DATASET DDL-NUMBER   6 11:43:46  4/ 1/75                         [0009] 000,0000492
*         ORDERING KEY MSFSET DDL-NUMBER  19 11:43:46  4/ 1/75
*            (SSNO).
*          02 SSNO                          PIC 9(9) COMP.                        [0010] 000,0000492
*          02 NONAM                         PIC 9 COMP.                                  000,0000501
*          02 LNAME                         PIC X(30).                            [0011] 000,0000502
*          02 MNAME                         PIC X(30).                                   000,0000562
*          02 FNAME                         PIC X(30).                                   000,0000622
*          02 CAMPUS-ADDRESS.                                                            000,0000682
*            03 DORM                          PIC X(6).                                  000,0000682
*            03 ROOM                          PIC 9(4) COMP.                             000,0000694
*            03 POBOX                         PIC 9(4) COMP.                             000,0000698
*            03 PHONE                         PIC 9(7) COMP.                             000,0000702
*          02 ND                            PIC 99 COMP.                                 000,0000710
*          02 DEGREE OCCURS 6 TIMES        PIC X(4).                                     000,0000712
*          02 TOTHRS                        PIC 999 COMP.                                000,0000760
*          02 TOTQP                         PIC 999 COMP.                                000,0000763
*          02 GRADE-POINT-AVG               PIC 9V99 COMP.                        [0012] 000,0000766
```

Figure H–3. COBOL Program Example (Sheet 1)

```
 *          02 MJR                          PIC 999 COMP.                                           000,0000769
 *          02 AMJR                         PIC X(18).                                              000,0000772
 *          02 SSEX                         PIC 9 COMP.                                             000,0000808
 *          02 SAGE                         PIC 99 COMP.                                            000,0000809
 *          02 HOME-ADDRESS SUBSET DDL-NUMBER  14 11:43:46  4/ 1/75 TO ADR
 *           DDL-NUMBER  10 11:43:46  4/ 1/75 .
 *          02 QUARTER DATASET DDL-NUMBER  15 11:43:46  4/ 1/75                               [0013] 000,0000812
 *           ORDERING KEY QSET DDL-NUMBER  15 11:43:46  4/ 1/75
 *              (QTR).
 *            03 QTR                          PIC X(4).                                       [0014] 000,0000812
 *            03 QTTRHRS                      PIC 99 COMP.                                           000,0000820
 *            03 QTRQP                        PIC 99 COMP.                                           000,0000822
 *            03 CORSES DATASET DDL-NUMBER  16 11:43:46  4/ 1/75                              [0015] 000,0000824
 *             ORDERING KEY CSET DDL-NUMBER  16 11:43:46  4/ 1/75
 *                (TYPECOURSE).
 *              04 TYPECOURSE                   PIC 9 COMP.                                          000,0000824
 *              04 GRADE                        PIC XX.                                              000,0000826
 *              04 CRS SUBSET DDL-NUMBER  17 11:43:46  4/ 1/75 TO
 *               UNIV-COURSES DDL-NUMBER   1 11:43:46  4/ 1/75 .
 *              04 YR                           PIC 99 COMP.                                         000,0000830
 *              04 Q                            PIC 99 COMP.                                         000,0000832
 *              04 GCRS SUBSET DDL-NUMBER  18 11:43:46  4/ 1/75 TO
 *               UNIV-COURSES DDL-NUMBER   1 11:43:46  4/ 1/75 .
 *              04 GGD                          PIC XX.                                              000,0000834
 *              04 TITLE-OF-PAPER               PIC X(30).                                           000,0000838
 *              04 PPRGD                        PIC XX.                                              000,0000898
001034    01 ADDRESS INVOKE ADR.
 *
 *
 *        01 ADR DATASET DDL-NUMBER  10 11:43:46  4/ 1/75                                     [0016] 000,0000902
 *         ORDERING KEY SAD DDL-NUMBER  20 11:43:46  4/ 1/75
 *            (ZIPC)
 *         ORDERING KEY SSAD DDL-NUMBER  21 11:43:46  4/ 1/75
 *            (SNO)
 *         ORDERING KEY STUAD DDL-NUMBER  22 11:43:46  4/ 1/75
 *            (ZIPC, SNO)
 *         ORDERING KEY FACAD DDL-NUMBER  23 11:43:46  4/ 1/75
 *            (ZIPC, SNO)
 *         ORDERING KEY ADMAD DDL-NUMBER  24 11:43:46  4/ 1/75
 *            (ZIPC, SNO).
 *          02 FACULTY-STUDENT              PIC 9 COMP.                                             000,0000902
 *          02 SNO                          PIC 9(9) COMP.                                          000,0000903
 *          02 ADLN OCCURS 9 TIMES         PIC X(54).                                               000,0000912
 *          02 ZIPC                         PIC 9(5) COMP.                                          000,0001884
 *          02 PHON                         PIC 9(10) COMP.                                         000,0001889
001035    WORKING-STORAGE SECTION.
001036        77  TOOMANYEXCEPTIONS   PIC 9(2) COMP.                                              [0017] 000,0001900
001037        77  TRUE                PIC 9 COMP VALUE "1".                                             000,0001902
001038    PROCEDURE DIVISION.
001044    BEGIN-SECTION SECTION.                                                                        000,0000000
001045    BEGIN.                                                                                        000,0000000
001046        OPEN OUTPUT MONITOR-DMS.                                                                  000,0000000
001047        MOVE SPACES TO MONITOR-REC.                                                               000,0000084
```

Figure H–3. COBOL Program Example (Sheet 2)

```
      001048        OPEN UPDATE UNIV ON EXCEPTION PERFORM STATUS-BOOLEAN.                                000,0000098
      002066    BUILD-MSF.                                                                               000,0000454
      002067        OPEN INPUT CARD.                                                                     000,0000454
      002068    READ-CARD-LOOP.                                                                          000,0000538
      002069        READ CARD AT END GO EOJ.                                                             000,0000538
      002070        WRITE MONITOR-REC FROM CARD-REC.                                                     000,0000788
      002071        IF C-TYPE = 1 GO 100-CREATE-MSF.                                                     000,0000982
      002072        IF C-TYPE = 2 GO 200-CREATE-QUARTER.                                                 000,0001018
      002073        IF C-TYPE = 3 GO 300-CREATE-CORSES.                                                  000,0001054
      002074                                                                                             000,0001090
      002075        IF C-TYPE = 4 GO 400-DELETE-ADR.                                                     000,0001090
      002076        IF C-TYPE = 5 GO 500-CHANGE-MSF-NAME.                                                000,0001126
      002077        IF C-TYPE = 6 GO 600-CHANGE-GRADE.                                                   000,0001162
      002078        DISPLAY C-TYPE "INVALID CARDTYPE" STOP RUN.                                          000,0001198
      002079    100-CREATE-MSF.                                                                          000,0001560
      002080        IF C-SSNO LESS THAN 1 OR GREATER THAN 10                                             000,0001560
      002081        MOVE "C-SSNO COLS 2-10 MUST BE BETWEEN 0 AND 11" TO                                  000,0001596
      002082        MONITOR-REC                                                                          000,0001636
      002083        WRITE MONITOR-REC GO READ-CARD-LOOP.                                                 000,0001636
      002084        CREATE MASTER ON EXCEPTION PERFORM STATUS-BOOLEAN.                                   000,0002160
      002085        MOVE C-SSNO TO SSNO.                                                                 000,0002446
      002086        MOVE C-GRD-PT-AVG TO GRADE-POINT-AVG.                                                000,0002460
      002087        IF C-SEX = "M" COMPUTE SSEX = TRUE.                                                  000,0002474
      002088        MOVE C-AGE TO SAGE.                                                                  000,0002600
      002089        MOVE C-NAME TO LNAME.                                                                000,0002662
      002090        STORE MASTER ON EXCEPTION PERFORM STATUS-BOOLEAN                                     000,0002676
      002091        GO EOJ.                                                                              000,0002936
      002092        GO TO READ-CARD-LOOP.                                                                000,0002980
      002093    200-CREATE-QUARTER.                                                                      000,0002998
      002094        MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                          000,0002998
      002095        IF DMSTATUS(NOTFOUND) DISPLAY C-SSNO "NOT IN MSF" ELSE                               000,0003344
      002096        PERFORM STATUS-BOOLEAN.                                                              000,0003700
      002097        CREATE QUARTER ON EXCEPTION PERFORM STATUS-BOOLEAN.                                  000,0003726
      002098        MOVE C-QTR TO QTR.                                                                   000,0004012
      002099        STORE QUARTER.                                                                       000,0004026
      002100        GO TO READ-CARD-LOOP.                                                                000,0004230
      003101    300-CREATE-CORSES.                                                                       000,0004248
      003102        MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                          000,0004248
      003103        IF DMSTATUS(NOTFOUND) DISPLAY C-SSNO " NOT IN MSF" ELSE                              000,0004594
      003104        PERFORM STATUS-BOOLEAN.                                                              000,0004958
      003105        MODIFY QSET AT QTR = C-QTR.                                                          000,0004984
WARNING...(254) SEQUENCE ERROR
                    CREATE CORSES.                                                                       000,0005274
      003108        MOVE C-TYPECOURSE TO TYPECOURSE.                                                     000,0005478
      003110        MOVE C-GRADE TO GGD.                                                                 000,0005540
      003111        STORE CORSES ON EXCEPTION PERFORM STATUS-BOOLEAN.                                    000,0005602
      003112        GO TO READ-CARD-LOOP.                                                                000,0005888
      003113    400-DELETE-ADR.                                                                          000,0005906
      003114        MOVE "MODIFY MSFSET     " TO MONITOR-VERB                                            000,0005906
      003115        MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                          000,0005906
      003116        PERFORM STATUS-BOOLEAN STOP RUN.                                                     000,0006410
WARNING...(254) SEQUENCE ERROR
                    MODIFY ADDRESS VIA FIRST HOME-ADDRESS ON EXCEPTION                                   000,0006476
```

Figure H–3. COBOL Program Example (Sheet 3)

```
                 IF DMSTATUS(NOTFOUND) GO TO READ-CARD-LOOP ELSE                                             000,0006784
                 PERFORM STATUS-BOOLEAN ELSE PERFORM REMOVE-ADDRESS.                                         000,0006848
     003117      MOVE "DELETE ADR        " TO MONITOR-VERB                                                   000,0006918
     003118      DELETE ADDRESS ON EXCEPTION PERFORM STATUS-BOOLEAN STOP RUN.                                000,0006918
     003119      GO TO READ-CARD-LOOP.                                                                       000,0007402
WARNING...(254) SEQUENCE ERROR
             REMOVE-ADDRESS.                                                                                 000,0007420
                 REMOVE CURRENT FROM HOME-ADDRESS ON EXCEPTION                                               000,0007420
                 PERFORM STATUS-BOOLEAN.                                                                     000,0007728
     003120  500-CHANGE-MSF-NAME.                                                                            000,0007754
     003121      MODIFY MSFSET AT SSNO = C-SSNO.                                                             000,0007771
     003122      MOVE LNAME TO MONITOR-REC. WRITE MONITOR-REC.                                               000,0008061
     003123      MOVE "NAME IN MSF WAS CHANGED TO" TO MONITOR-REC.                                           000,0008231
     003124      WRITE MONITOR-REC.                                                                          000,0008461
     003125      MOVE C-NAME TO LNAME.                                                                       000,0008617
     003126      STORE MASTER.                                                                               000,0008631
     003127      MOVE LNAME TO MONITOR-REC. WRITE MONITOR-REC.                                               000,0008835
     003128      GO TO READ-CARD-LOOP.                                                                       000,0009005
     003129  600-CHANGE-GRADE.                                                                               000,0009023
     003130      MOVE "MODIFY MSFSET     " TO MONITOR-VERB.                                                  000,0009023
     003131      MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                                 000,0009181
     003132      PERFORM STATUS-BOOLEAN.                                                                     000,0009527
     003133      MOVE C-GRD-PT-AVG TO GRADE-POINT-AVG.                                                       000,0009553
     003134      MOVE "STORE MSF         " TO MONITOR-VERB.                                                  000,0009567
     003135      STORE MASTER ON EXCEPTION PERFORM STATUS-BOOLEAN.                                           000,0009725
     003136      GO TO READ-CARD-LOOP.                                                                       000,0010011
     003137  STATUS-BOOLEAN.                                                                                 000,0010029
     003138      ADD 1 TO TOOMANYEXCEPTIONS.                                                                 000,0010029
     003139      IF TOOMANYEXCEPTIONS GREATER THAN 10                                                        000,0010043
     003140      DISPLAY TOOMANYEXCEPTIONS "IS TOO MANY EXCEPTIONS" STOP RUN.                                000,0010043
     003141      MOVE ALL "*" TO MONITOR-EXCEPTION.                                                          000,0010493
     003142      IF DMSTATUS (NOTFOUND    ) THEN MOVE "NOT FOUND     "                                       000,0010617
     003143      TO MONITOR-STATUS-B.                                                                        000,0010681
     003144      IF DMSTATUS (DUPLICATES ) THEN MOVE "DUPLICATES    "                                        000,0010807
     003145      TO MONITOR-STATUS-B.                                                                        000,0010871
     003146      IF DMSTATUS (DEADLOCK    ) THEN MOVE "DEADLOCK      "                                       000,0010997
     003147      TO MONITOR-STATUS-B.                                                                        000,0011061
     003148      IF DMSTATUS (DATAERROR  ) THEN MOVE "DATA ERROR    "                                        000,0011187
     003149      TO MONITOR-STATUS-B.                                                                        000,0011251
     003150      IF DMSTATUS (NOTLOCKED  ) THEN MOVE "NOT LOCKED    "                                        000,0011377
     003151      TO MONITOR-STATUS-B.                                                                        000,0011441
     003152      IF DMSTATUS (KEYCHANGED ) THEN MOVE "KEY CHANGED   "                                        000,0011567
     003153      TO MONITOR-STATUS-B.                                                                        000,0011631
     003154      IF DMSTATUS (SYSTEMERROR) THEN MOVE "SYSTEM ERROR  "                                        000,0011757
     003155      TO MONITOR-STATUS-B.                                                                        000,0011821
     003156      IF DMSTATUS (IOERROR     ) THEN MOVE "IO ERROR      "                                       000,0011947
     003157      TO MONITOR-STATUS-B.                                                                        000,0012011
     003158      IF DMSTATUS (LIMITERROR ) THEN MOVE "LIMIT ERROR   "                                        000,0012137
     003159      TO MONITOR-STATUS-B.                                                                        000,0012201
     003160      IF DMSTATUS (OPENERROR  ) THEN MOVE "OPEN ERROR    "                                        000,0012327
     004161      TO MONITOR-STATUS-B.                                                                        000,0012391
     004162      IF DMSTATUS (CLOSEERROR ) THEN MOVE "CLOSE ERROR   "                                        000,0012517
     004163      TO MONITOR-STATUS-B.                                                                        000,0012581
```

Figure H–3. COBOL Program Example (Sheet 4)

```
004164       IF DMSTATUS (NORECORD    ) THEN MOVE "NO RECORD      "                                                    000,0012707
004165       TO MONITOR-STATUS-B.                                                                                    000,0012771
004166       IF DMSTATUS (INUSE       ) THEN MOVE "IN USE         "                                                    000,0012697
004167       TO MONITOR-STATUS-B.                                                                                    000,0012961
004168       WRITE MONITOR-REC.                                                                                      000,0013087
004169       MOVE SPACES TO MONITOR-REC.                                                                             000,0013243
004170   EOJ.                                                                                                        000,0013257
004171       IF TOOMANYEXCEPTIONS = 0 DISPLAY "NO DM EXCEPTION".                                                     000,0013274
004172   CLOSE UNIV ON EXCEPTION STOP RUN.                                                                           000,0013604
004173   STOP RUN.                                                                                                   000,0013974
999999   END-OF-JOB.                                                                                                 000,0014014



C  O  D  E     D  I  C  T  I  O  N  A  R  Y
    BYTE LENGTH   CODEFILE RELATIVE DISK ADR

    ...........   .........................
000 00001752      000011
    00001752      TOTAL CODE

D  A  T  A     D  I  C  T  I  O  N  A  R  Y
    BYTE LENGTH   CODEFILE RELATIVE DISK ADR

    ...........   .........................
000 00001211      000003

P  A  T  H     D  I  C  T  I  O  N  A  R  Y
    INVOKED   STRUCTURE#  HH:MM:SS   MM/DD/YY

    .......   ..........  ........   ........
000 FALSE     0000        00:00:00   00/00/00
001 TRUE      0006        11:43:46   04/01/75
002 FALSE     0019        11:43:46   04/01/75
003 FALSE     0014        11:43:46   04/01/75
004 TRUE      0010        11:43:46   04/01/75
005 FALSE     0015        11:43:46   04/01/75
006 FALSE     0016        11:43:46   04/01/75
007 FALSE     0017        11:43:46   04/01/75
008 TRUE      0001        11:43:46   04/01/75
009 FALSE     0018        11:43:46   04/01/75
010 FALSE     0020        11:43:46   04/01/75
011 FALSE     0021        11:43:46   04/01/75
012 FALSE     0022        11:43:46   04/01/75
013 FALSE     0023        11:43:46   04/01/75
014 FALSE     0024        11:43:46   04/01/75

S - M  A  C  H  I  N  E     P  A  R  A  M  E  T  E  R  S     ( S  C  R  A  T  C  H  P  A  D )

LENB=9, SEGB=0, DISPB=11, COPXB=5, COPB=24, D.E.F=354, BDISPB=14
BASE RELATIVE ADDRESSES:
  DATA-SEGMENT-0=616, COP-TABLE=40, STACK=8681 (BIT LENGTH=1000)
```

Figure H–3. COBOL Program Example (Sheet 5)

```
 P R O G R A M   P A R A M E T E R   B L O C K

 FIRST-EXECUTABLE-INSTRUCTION=0,0
 INTERPRETER-NAME=COBOL     /INTERP
 STATIC-CORE=9688 BITS
 DYNAMIC-CORE=0 BITS
 DATA DICTIONARY STARTS AT CODEFILE SEGMENT 2, 1 ENTRY
 CODE DICTIONARY STARTS AT CODEFILE SEGMENT 10, 1 ENTRY
 FILE PARAMETER BLOCKS START AT CODEFILE SEGMENT 21, 2 ENTRIES
 PATH DICTIONARY STARTS AT CODEFILE SEGMENT 23, 15 ENTRIES

LAST ERROR AT SEQUENCE NUMBER        . 3 WARNINGS 3 SEQUENCE ERRORS.
***** COMPILATION COMPLETE
ELAPSED TIME : 01 MINUTE, 46 SECONDS
PROGRAM REQUIRES 23 DISK SEGMENTS OF 180 BYTES EACH.
MEMORY REQUIREMENTS
  0001752 BYTES = LARGEST CODE SEGMENT
  0001211 BYTES = BASE-TO-LIMIT AREA
  0000315 BYTES = DICTIONARIES AND RUN STRUCTURE
  0000505 BYTES = FILE BUFFERS & FILE INFO AREAS - INCLUDES 129 BYTES (+72 TO 540 IF DISK) FOR EACH FILE
  0003783 BYTES = ESTIMATED MEMORY REQUIRED TO RUN IF ALL FILES OPEN
221 SYMBOLIC RECORDS COMPILED AT 125,040 RECORDS PER MINUTE.


 001001  IDENTIFICATION DIVISION.
 001002  PROGRAM-ID. DMSCOBOLSAMPLE.
 001003  ENVIRONMENT DIVISION.
 001004  INPUT-OUTPUT SECTION.
 001005  FILE-CONTROL.
 001006      SELECT CARD ASSIGN TO READER.
 001007      SELECT MONITOR-DMS ASSIGN TO PRINTER.
 001008  DATA DIVISION.
 001009  FILE SECTION.
 001010  FD  CARD.
 001011  01  CARD-REC.                                                                   000,0000066
 001012      03  C-TYPE          PIC 9.                                         [0001] 000,0000066
 001013      03  C-SSNO          PIC 9(9).                                      [0002] 000,0000068
 001014      03  C-GRD-PT-AVG    PIC 999V99.                                    [0003] 000,0000086
 001015      03  C-SEX           PIC X.                                                000,0000096
 001016      03  C-AGE           PIC 99.                                               000,0000098
 001017      03  C-QTR           PIC X(4).                                      [0004] 000,0000102
 001018      03  C-TYPECOURSE    PIC 9.                                                000,0000110
 001019      03  C-GRADE         PIC XX.                                               000,0000112
 001020      03  C-TITLE-PAPER   PIC X(30).                                            000,0000116
 001021      03  C-NAME          PIC X(24).                                     [0005] 000,0000176
 001022  FD  MONITOR-DMS.
 001023  01  MONITOR-REC.                                                       [0006] 000,0000224
 001024      03  MONITOR-EXCEPTION   PIC X(4).                                         000,0000224
 001025      03  MONITOR-STATUS-B    PIC X(20).                                 [0007] 000,0000232
 001026      03  MONITOR-STATUS      PIC ZZZ9BB.                                       000,0000272
 001027      03  MONITOR-VERB        PIC X(20).                                 [0008] 000,0000284
 001028      03  MONITOR-SET         PIC X(17).                                        000,0000324
```

Figure H–3. COBOL Program Example (Sheet 6)

```
001029       03  MONITOR-STRUCTURE   PIC 9(3).                                              000,0000358
001030       03  FILLER              PIC X(64).                                             000,0000364
001031   DATA-BASE SECTION.
001032   DB  UNIV.
001033   01 MASTER INVOKE MSF.
*
*
*        01 MSF DATASET DDL-NUMBER   6 11:43:46  4/ 1/75                            [0009] 000,0000492
*         ORDERING KEY MSFSET DDL-NUMBER  19 11:43:46  4/ 1/75
*            (SSNO).
*          02 SSNO                          PIC 9(9) COMP.                          [0010] 000,0000492
*          02 NONAM                         PIC 9 COMP.                                    000,0000501
*          02 LNAME                         PIC X(30).                              [0011] 000,0000502
*          02 MNAME                         PIC X(30).                                     000,0000562
*          02 FNAME                         PIC X(30).                                     000,0000622
*          02 CAMPUS-ADDRESS.                                                              000,0000682
*            03 DORM                          PIC X(6).                                    000,0000682
*            03 ROOM                          PIC 9(4) COMP.                               000,0000694
*            03 POBOX                         PIC 9(4) COMP.                               000,0000698
*            03 PHONE                         PIC 9(7) COMP.                               000,0000702
*          02 ND                            PIC 99 COMP.                                   000,0000710
*          02 DEGREE OCCURS 6 TIMES        PIC X(4).                                       000,0000712
*          02 TOTHRS                        PIC 999 COMP.                                  000,0000760
*          02 TOTQP                         PIC 999 COMP.                                  000,0000763
*          02 GRADE-POINT-AVG               PIC 9V99 COMP.                          [0012] 000,0000766
*          02 MJR                           PIC 999 COMP.                                  000,0000769
*          02 AMJR                          PIC X(18).                                     000,0000772
*          02 SSEX                          PIC 9 COMP.                                    000,0000808
*          02 SAGE                          PIC 99 COMP.                                   000,0000809
*          02 HOME-ADDRESS SUBSET DDL-NUMBER  14 11:43:46  4/ 1/75 TO ADR
*           DDL-NUMBER  10 11:43:46  4/ 1/75 .
*          02 QUARTER DATASET DDL-NUMBER  15 11:43:46  4/ 1/75                      [0013] 000,0000812
*           ORDERING KEY QSET DDL-NUMBER  15 11:43:46  4/ 1/75
*              (QTR).
*            03 QTR                           PIC X(4).                             [0014] 000,0000812
*            03 QTTRHRS                       PIC 99 COMP.                                 000,0000820
*            03 QTRQP                         PIC 99 COMP.                                 000,0000822
*            03 CORSES DATASET DDL-NUMBER  16 11:43:46  4/ 1/75                     [0015] 000,0000824
*             ORDERING KEY CSET DDL-NUMBER  16 11:43:46  4/ 1/75
*                (TYPECOURSE).
*              04 TYPECOURSE                    PIC 9 COMP.                                000,0000824
*              04 GRADE                         PIC XX.                                    000,0000826
*              04 CRS SUBSET DDL-NUMBER  17 11:43:46  4/ 1/75 TO
*               UNIV-COURSES DDL-NUMBER   1 11:43:46  4/ 1/75 .
*              04 YR                            PIC 99 COMP.                               000,0000830
*              04 Q                             PIC 99 COMP.                               000,0000832
*              04 GCRS SUBSET DDL-NUMBER  18 11:43:46  4/ 1/75 TO
*               UNIV-COURSES DDL-NUMBER   1 11:43:46  4/ 1/75 .
*              04 GGD                           PIC XX.                                    000,0000834
*              04 TITLE-OF-PAPER                PIC X(30).                                 000,0000838
*              04 PPRGD                         PIC XX.                                    000,0000898
001034   01 ADDRESS INVOKE ADR.
*
```

Figure H–3. COBOL Program Example (Sheet 7)

```
*
*         01 ADR DATASET DDL=NUMBER  10 11:43:46  4/ 1/75                                [0016] 000,0000902
*          ORDERING KEY SAD DDL=NUMBER  20 11:43:46  4/ 1/75
*             (ZIPC)
*          ORDERING KEY SSAD DDL=NUMBER  21 11:43:46  4/ 1/75
*             (SNO)
*          ORDERING KEY STUAD DDL=NUMBER  22 11:43:46  4/ 1/75
*             (ZIPC, SNO)
*          ORDERING KEY FACAD DDL=NUMBER  23 11:43:46  4/ 1/75
*             (ZIPC, SNO)
*          ORDERING KEY ADMAD DDL=NUMBER  24 11:43:46  4/ 1/75
*             (ZIPC, SNO).
*           02 FACULTY=STUDENT                 PIC 9 COMP.                                    000,0000902
*           02 SNO                             PIC 9(9) COMP.                                 000,0000903
*           02 ADLN OCCURS 9 TIMES            PIC X(54).                                      000,0000912
*           02 ZIPC                            PIC 9(5) COMP.                                 000,0001884
*           02 PHON                            PIC 9(10) COMP.                                000,0001889
 001035   WORKING=STORAGE SECTION.
 001036       77  TOOMANYEXCEPTIONS    PIC 9(2) COMP.                                    [0017] 000,0001900
 001037       77  TRUE                 PIC 9 COMP VALUE "1".                                    000,0001902
 001038   PROCEDURE DIVISION.
 001044   BEGIN=SECTION SECTION.                                                                 000,0000000
 001045   BEGIN.                                                                                 000,0000000
 001046       OPEN OUTPUT MONITOR=DMS.                                                           000,0000000
 001047       MOVE SPACES TO MONITOR=REC.                                                        000,0000084
 001048       OPEN UPDATE UNIV ON EXCEPTION PERFORM STATUS=BOOLEAN.                              000,0000098
 002066   BUILD=MSF.                                                                             000,0000454
 002067       OPEN INPUT CARD.                                                                   000,0000454
 002068   READ=CARD=LOOP.                                                                        000,0000538
 002069       READ CARD AT END GO EOJ.                                                           000,0000538
 002070       WRITE MONITOR=REC FROM CARD=REC.                                                   000,0000788
 002071       IF C=TYPE = 1 GO 100=CREATE=MSF.                                                   000,0000982
 002072       IF C=TYPE = 2 GO 200=CREATE=QUARTER.                                               000,0001018
 002073       IF C=TYPE = 3 GO 300=CREATE=CORSES.                                                000,0001054
 002074                                                                                          000,0001090
 002075       IF C=TYPE = 4 GO 400=DELETE=ADR.                                                   000,0001090
 002076       IF C=TYPE = 5 GO 500=CHANGE=MSF=NAME.                                              000,0001126
 002077       IF C=TYPE = 6 GO 600=CHANGE=GRADE.                                                 000,0001162
 002078       DISPLAY C=TYPE "INVALID CARDTYPE" STOP RUN.                                        000,0001198
 002079   100=CREATE=MSF.                                                                        000,0001560
 002080       IF C=SSNO LESS THAN 1 OR GREATER THAN 10                                           000,0001560
 002081       MOVE "C=SSNO COLS 2=10 MUST BE BETWEEN 0 AND 11" TO                                000,0001596
 002082       MONITOR=REC                                                                        000,0001636
 002083       WRITE MONITOR=REC GO READ=CARD=LOOP.                                               000,0001636
 002084       CREATE MASTER ON EXCEPTION PERFORM STATUS=BOOLEAN.                                 000,0002160
 002085       MOVE C=SSNO TO SSNO.                                                               000,0002446
 002086       MOVE C=GRD=PT=AVG TO GRADE=POINT=AVG.                                              000,0002460
 002087       IF C=SEX = "M" COMPUTE SSEX = TRUE.                                                000,0002474
 002088       MOVE C=AGE TO SAGE.                                                                000,0002600
 002089       MOVE C=NAME TO LNAME.                                                              000,0002662
 002090       STORE MASTER ON EXCEPTION PERFORM STATUS=BOOLEAN                                   000,0002676
 002092       GO TO READ=CARD=LOOP.                                                              000,0002936
```

Figure H–3. COBOL Program Example (Sheet 8)

```
      002093   200-CREATE-QUARTER.                                                                                    000,0002980
      002094       MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                                        000,0002980
      002095       IF DMSTATUS(NOTFOUND) DISPLAY C-SSNO "NOT IN MSF" ELSE                                             000,0003326
      002096       PERFORM STATUS-BOOLEAN.                                                                            000,0003682
      002097       CREATE QUARTER ON EXCEPTION PERFORM STATUS-BOOLEAN.                                                000,0003708
      002098       MOVE C-QTR TO QTR.                                                                                 000,0003994
      002099       STORE QUARTER.                                                                                     000,0004008
      002100       GO TO READ-CARD-LOOP.                                                                              000,0004212
      003101   300-CREATE-CORSES.                                                                                     000,0004230
      003102       MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                                        000,0004230
      003103       IF DMSTATUS(NOTFOUND) DISPLAY C-SSNO " NOT IN MSF" ELSE                                            000,0004576
      003104       PERFORM STATUS-BOOLEAN.                                                                            000,0004940
      003105       MODIFY QSET AT QTR = C-QTR.                                                                        000,0004966
WARNING...(254) SEQUENCE ERROR
                   CREATE CORSES.                                                                                     000,0005256
      003108       MOVE C-TYPECOURSE TO TYPECOURSE.                                                                   000,0005460
      003110       MOVE C-GRADE TO GGD.                                                                               000,0005522
      003111       STORE CORSES ON EXCEPTION PERFORM STATUS-BOOLEAN.                                                  000,0005584
      003112       GO TO READ-CARD-LOOP.                                                                              000,0005870
      003113   400-DELETE-ADR.                                                                                        000,0005888
      003114       MOVE "MODIFY MSFSET    " TO MONITOR-VERB                                                           000,0005888
      003115       MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                                        000,0005888
      003116       PERFORM STATUS-BOOLEAN STOP RUN.                                                                   000,0006392
WARNING...(254) SEQUENCE ERROR
                   MODIFY ADDRESS VIA FIRST HOME-ADDRESS ON EXCEPTION                                                 000,0006458
                   IF DMSTATUS(NOTFOUND) GO TO READ-CARD-LOOP ELSE                                                    000,0006766
                   PERFORM STATUS-BOOLEAN ELSE PERFORM REMOVE-ADDRESS.                                                000,0006830
      003117       MOVE "DELETE ADR       " TO MONITOR-VERB                                                           000,0006900
      003118       DELETE ADDRESS ON EXCEPTION PERFORM STATUS-BOOLEAN STOP RUN.                                       000,0006900
      003119       GO TO READ-CARD-LOOP.                                                                              000,0007384
WARNING...(254) SEQUENCE ERROR
               REMOVE-ADDRESS.                                                                                        000,0007402
                   REMOVE CURRENT FROM HOME-ADDRESS ON EXCEPTION                                                      000,0007402
                   PERFORM STATUS-BOOLEAN.                                                                            000,0007710
      003120   500-CHANGE-MSF-NAME.                                                                                   000,0007736
      003121       MODIFY MSFSET AT SSNO = C-SSNO.                                                                    000,0007753
      003122       MOVE LNAME TO MONITOR-REC. WRITE MONITOR-REC.                                                      000,0008043
      003123       MOVE "NAME IN MSF WAS CHANGED TO" TO MONITOR-REC.                                                  000,0008213
      003124       WRITE MONITOR-REC.                                                                                 000,0008443
      003125       MOVE C-NAME TO LNAME.                                                                              000,0008599
      003126       STORE MASTER.                                                                                      000,0008613
      003127       MOVE LNAME TO MONITOR-REC. WRITE MONITOR-REC.                                                      000,0008817
      003128       GO TO READ-CARD-LOOP.                                                                              000,0008987
      003129   600-CHANGE-GRADE.                                                                                      000,0009005
      003130       MOVE "MODIFY MSFSET    " TO MONITOR-VERB.                                                          000,0009005
      003131       MODIFY MSFSET AT SSNO = C-SSNO ON EXCEPTION                                                        000,0009163
      003132       PERFORM STATUS-BOOLEAN.                                                                            000,0009509
      003133       MOVE C-GRD-PT-AVG TO GRADE-POINT-AVG.                                                              000,0009535
      003134       MOVE "STORE MSF        " TO MONITOR-VERB.                                                          000,0009549
      003135       STORE MASTER ON EXCEPTION PERFORM STATUS-BOOLEAN.                                                  000,0009707
      003136       GO TO READ-CARD-LOOP.                                                                              000,0009993
      003137   STATUS-BOOLEAN.                                                                                        000,0010011
      003138       ADD 1 TO TOOMANYEXCEPTIONS.                                                                        000,0010011
```

Figure H–3. COBOL Program Example (Sheet 9)

```
003139       IF TOOMANYEXCEPTIONS GREATER THAN 10                                                000,0010025
003140       DISPLAY TOOMANYEXCEPTIONS "IS TOO MANY EXCEPTIONS" STOP RUN.                        000,0010025
003141       MOVE ALL "*" TO MONITOR-EXCEPTION.                                                  000,0010475
003142       IF DMSTATUS (NOTFOUND   ) THEN MOVE "NOT FOUND     "                                 000,0010599
003143       TO MONITOR-STATUS-B.                                                                000,0010663
003144       IF DMSTATUS (DUPLICATES ) THEN MOVE "DUPLICATES    "                                 000,0010789
003145       TO MONITOR-STATUS-B.                                                                000,0010853
003146       IF DMSTATUS (DEADLOCK   ) THEN MOVE "DEADLOCK      "                                 000,0010979
003147       TO MONITOR-STATUS-B.                                                                000,0011043
003148       IF DMSTATUS (DATAERROR  ) THEN MOVE "DATA ERROR    "                                 000,0011169
003149       TO MONITOR-STATUS-B.                                                                000,0011233
003150       IF DMSTATUS (NOTLOCKED  ) THEN MOVE "NOT LOCKED    "                                 000,0011359
003151       TO MONITOR-STATUS-B.                                                                000,0011423
003152       IF DMSTATUS (KEYCHANGED ) THEN MOVE "KEY CHANGED   "                                 000,0011549
003153       TO MONITOR-STATUS-B.                                                                000,0011613
003154       IF DMSTATUS (SYSTEMERROR) THEN MOVE "SYSTEM ERROR  "                                 000,0011739
003155       TO MONITOR-STATUS-B.                                                                000,0011803
003156       IF DMSTATUS (IOERROR    ) THEN MOVE "IO ERROR      "                                 000,0011929
003157       TO MONITOR-STATUS-B.                                                                000,0011993
003158       IF DMSTATUS (LIMITERROR ) THEN MOVE "LIMIT ERROR   "                                 000,0012119
003159       TO MONITOR-STATUS-B.                                                                000,0012183
003160       IF DMSTATUS (OPENERROR  ) THEN MOVE "OPEN ERROR    "                                 000,0012309
004161       TO MONITOR-STATUS-B.                                                                000,0012373
004162       IF DMSTATUS (CLOSEERROR ) THEN MOVE "CLOSE ERROR   "                                 000,0012499
004163       TO MONITOR-STATUS-B.                                                                000,0012563
004164       IF DMSTATUS (NORECORD   ) THEN MOVE "NO RECORD     "                                 000,0012689
004165       TO MONITOR-STATUS-B.                                                                000,0012753
004166       IF DMSTATUS (INUSE      ) THEN MOVE "IN USE        "                                 000,0012879
004167       TO MONITOR-STATUS-B.                                                                000,0012943
004168       WRITE MONITOR-REC.                                                                  000,0013069
004169       MOVE SPACES TO MONITOR-REC.                                                         000,0013225
004170   EOJ.                                                                                    000,0013239
004171       IF TOOMANYEXCEPTIONS = 0 DISPLAY "NO DM EXCEPTION".                                 000,0013256
004172   CLOSE UNIV ON EXCEPTION STOP RUN.                                                       000,0013586
004173   STOP RUN.                                                                               000,0013956
999999   END-OF-JOB.                                                                             000.0013996



C. O D E     D I C T I O N A R Y
    BYTE LENGTH   CODEFILE RELATIVE DISK ADR

    ...........   .........................
000 00001750      000011
    00001750      TOTAL CODE

D A T A      D I C T I O N A R Y
    BYTE LENGTH   CODEFILE RELATIVE DISK ADR

    ...........   .........................
000 00001211      000003
```

Figure H–3. COBOL Program Example (Sheet 10)

```
P  A  T  H    D  I  C  T  I  O  N  A  R  Y
    INVOKED    STRUCTURE#  HH:MM:SS   MM/DD/YY
    .......    ..........  ........   ........
000 FALSE      0000        00:00:00   00/00/00
001 TRUE       0006        11:43:46   04/01/75
002 FALSE      0019        11:43:46   04/01/75
003 FALSE      0014        11:43:46   04/01/75
004 TRUE       0010        11:43:46   04/01/75
005 FALSE      0015        11:43:46   04/01/75
006 FALSE      0016        11:43:46   04/01/75
007 FALSE      0017        11:43:46   04/01/75
008 TRUE       0001        11:43:46   04/01/75
009 FALSE      0018        11:43:46   04/01/75
010 FALSE      0020        11:43:46   04/01/75
011 FALSE      0021        11:43:46   04/01/75
012 FALSE      0022        11:43:46   04/01/75
013 FALSE      0023        11:43:46   04/01/75
014 FALSE      0024        11:43:46   04/01/75

S - M  A  C  H  I  N  E    P  A  R  A  M  E  T  E  R  S    ( S  C  R  A  T  C  H  P  A  D )

LENB=9, SEGB=0, DISPB=11, COPXB=5, COPB=24, D.E.F=354, BDISPB=14
BASE RELATIVE ADDRESSES:
  DATA-SEGMENT-0=616, COP-TABLE=40, STACK=8681 (BIT LENGTH=1000)

P  R  O  G  R  A  M    P  A  R  A  M  E  T  E  R    B  L  O  C  K

FIRST-EXECUTABLE-INSTRUCTION=0,0
INTERPRETER-NAME=COBOL      /INTERP
STATIC-CORE=9688 BITS
DYNAMIC-CORE=0 BITS
DATA DICTIONARY STARTS AT CODEFILE SEGMENT 2, 1 ENTRY
CODE DICTIONARY STARTS AT CODEFILE SEGMENT 10, 1 ENTRY
FILE PARAMETER BLOCKS START AT CODEFILE SEGMENT 21, 2 ENTRIES
PATH DICTIONARY STARTS AT CODEFILE SEGMENT 23, 15 ENTRIES

LAST ERROR AT SEQUENCE NUMBER        . 3 WARNINGS 3 SEQUENCE ERRORS.
***** COMPILATION COMPLETE
ELAPSED TIME : 01 MINUTE, 58 SECONDS
PROGRAM REQUIRES 23 DISK SEGMENTS OF 180 BYTES EACH.
MEMORY REQUIREMENTS
  0001750 BYTES = LARGEST CODE SEGMENT
  0001211 BYTES = BASE-TO-LIMIT AREA
  0000315 BYTES = DICTIONARIES AND RUN STRUCTURE
  0000505 BYTES = FILE BUFFERS & FILE INFO AREAS - INCLUDES 129 BYTES (+72 TO 540 IF DISK) FOR EACH FILE
  0003781 BYTES = ESTIMATED MEMORY REQUIRED TO RUN IF ALL FILES OPEN
220 SYMBOLIC RECORDS COMPILED AT 111,840 RECORDS PER MINUTE.
```

Figure H–3. COBOL Program Example (Sheet 11)

UNIV DATA BASE IDENTIFIERS

ADDRES	Points to the ADR data set which contains the address of a PROFESSOR.
ADLN	A record (ALPHA data item) in ADR data set which may contain up to nine lines of addresses.
ADMAD	A subset pointing to the records in ADR that are administrators, and are arranged by zip code, social security number sequence.
ADR	A common address file containing address records of students, professors, and administrators.
AGE	A NUMERIC data item which contains the age of university personnel.
AMJR	An ALPHA data item which contains the name of the subject a student is taking as a major.
AUTHR	An ALPHA data item that contains the name of an author of a book which is used in a course.
BOOKS	An embedded data set. Since the quantity of books used in a course may vary, an embedded set is defined to avoid specifying one occurrence of a field that occurs several times. An embedded UNORDERED data set is useful when the number of records per parent record is small. In this case, the parent record is a record in UNIV-COURSES. Most courses never use more than two or three books. An exception to this would be an English literature course.
BUILDING	A NUMERIC data item which identifies the building on campus where a specific course is taught.
CAMPUS-ADDRESS	A GROUP data item containing both ALPHA and NUMERIC information of a student's address.
CLASS-SIZE	A NUMERIC data item with a field length of two digits which specifies the number of students currently enrolled in a course.
CORSES	An embedded data set within QUARTER data set. Refer to QUARTER. Contains records of courses completed during any given quarter by a student.
COURSENAME	The name of a course.
COURSES	COURSES points to UNIV-COURSES data set, and indicates the courses that are taught by a professor. A professor normally teaches a maximum of eight different courses.
CRS-NAME	A GROUP level item which identifies a record of the UNIV-COURSES data set.
CRS-NO	The symbolic key used to retrieve a record in the UNIV-COURSES data set. CRS-NO contains the number that has been arbitrarily assigned to a course.

CSET	Enables the retrieval of a record from the CORSES data set.
DAYS-OF-WEEK	A string of digits indicating the days of the week a course is being taught. Each digit in the string has a unique name that can be referenced as a single-digit number.
DEGREE	An ALPHA data item indicating the number of degrees (maximum limit is 6) a student may have previously earned. Refer to the NUMERIC data item ND.
DEPARTMENT	An ALPHA data item indicating the department within the university in the UNIV-COURSES data set.
DORM	An ALPHA data item which is part of a student's campus address.
DPT	A data item within the UNIV-PERSONNEL data set. This data item defines the department of which university personnel are a part (Science, Mathematics, or Foreign Language).
FACAD	A subset pointing to the records in ADR that are professors, and are arranged by zip code, social security number sequence.
FACULTY-STUDENT	Denotes the type of address record in the ADR data set as follows: 1. Indicates a student. 2. Indicates a professor. 3. Indicates an administrative person.
FIRSTNAME	An ALPHA data item in the UNIV-PERSONNEL data set which contains the first name of a professor. This data item can have a maximum length of 10 characters.
FLAG-BITS	A string of digits that is currently undefined within the data base. In the future, the university may want to establish some FLAG FIELDS and, at that time, this space could be used.
FNAME	An ALPHA data item containing the first name of a student, and which is specified to be not greater than 30 characters in length.
FRI	Indicates the course is offered on Friday.
GCRS	Points to a record in the UNIV-COURSES data set.
GGD	Grade received for an Undergraduate course.
GRADE-POINT-AVG	Self-explanatory.
HOME-ADDRESS	Points to a record in the ADR data set. See ADR.
HOURSCRDT	The number of hours of credit that can be earned by successfully completing this course.
LASTNAME	The last name of a professor, and is an ALPHA data item which is in the UNIV-PERSONNEL data set.

LC	Library of Congress number for a book used in a course.
LEVEL	A numeric item indicating the level of a course, e.g., Graduate, Undergraduate, Advanced, or Elementary.
LNAME	Last name of a student.
MON	Indicates the course is offered on Monday.
MSF	A master file of students.
MSFSET	Retrieve a record from the MSF data set, using symbolic key of SSNO (Social Security Number). See SSNO.
NAME	This is a GROUP item. It is the name of a person working for the university. It is also a key of retrieval. See U-P-SET.
ND	Number of degrees previously earned by a student. See DEGREE.
NONAM	Number of middle names for a student. (Only the first one is carried in the data base.)
NOPROF	The number of different professors that teach a given course.
PHON	Phone number at HOME-ADDRESS.
PHONE	Phone number at student's CAMPUS-ADDRESS.
POBOX	Post office box (mail box) of student's CAMPUS-ADDRESS.
PPRGD	The grade the student earned on a paper written for a graduate course.
PROFESSOR	This points to the professor who teaches this course. There will typically be a maximum of three professors teaching this course.
Q	Quality points which are assigned to graduate course. Quality points may be different depending on the student.
QSET	Retrieve a record from QUARTER data set.
QTR	Identifies the quarter. For example, SU72 would be summer of 1972.
QTRQP	The total quality points earned by formula=grade X HOURSCRDT. Grade must be converted to numeric value first.
QTTRHRS	The total credit hours successfully completed by student during a particular quarter.
QUARTER	An embedded data set embedded within MSF. Contains a record for each quarter that a student has attended the university.
RANK	Associate professor, full professor, or department head.
ROOM	Part of student's CAMPUS-ADDRESS.

ROOM NUMBER	This gives the location of where a course is taught (e.g., which room in a building). See BUILDING.
SAGE	Student's age.
SALARY	Normal remuneration.
SAT	Indicates course is offered on Saturday.
SEX	Male or female.
SNO	Social Security of student administration or faculty number.
SSAD	Retrieve a record in ADR data set using symbolic key of SNO. See SNO.
SSEX	Male or female student. 1 if male student; 0 if female.
SSNO	Social Security Number of a student.
SSNUM	Social Security Number. See also SS-U-P.
SS-U-P	Retrieve a record from UNIV-PERSONNEL using symbolic key of SSNUM (Social Security Number). See SSNUM.
STUAD	A subset pointing to the records in ADR that are students, and are arranged by zip code, social security number sequence.
SUPR	Points to supervisor (who is also a professor) of this professor.
THURS	Indicates course is offered on Thursday.
TITLES	The title of a book used in a course.
TITLE-OF-PAPER	Descriptive title of paper written by a student for a graduate course.
TOTHRS	The total credit hours student has attended.
TOTQP	The total quality points earned by a student.
TUES	Indicates course is offered on Tuesday.
TYPECOURSE	Symbolic key of CORSES data set. See CSET.
UNIV-COURSES	The courses offered by this university.
UNIV-C-SET	Retrieve a record from UNIV-COURSES using CRS-NO as a symbolic key.
UNIV-PERSONNEL	Data set containing a record for each person working for the university.
U-P-SET	Retrieve a record from UNIV-PERSONNEL using symbolic key of NAME. See NAME.

WEDS	Indicates the course is offered on Wednesday.
YR	The year a particular graduate course was taken.
ZIPC	Zip code for ADR.

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: B 1700 SYSTEMS DATA MANAGEMENT SYSTEM II (DMS II) Reference Manual

FORM: 1089794
DATE: January, 1976

CHECK TYPE OF SUGGESTION:

☐ ADDITION ☐ DELETION ☐ REVISION ☐ ERROR

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME ____________________ DATE ____________
TITLE ____________________
COMPANY ____________________
ADDRESS ____________________
____

cut along dotted line

cut along dotted line

STAPLE

FOLD DOWN SECOND FOLD DOWN

Postage Will Be Paid by Addressee

No Postage Stamp Necessary If Mailed in the United States

BUSINESS REPLY MAIL
First Class Permit No. 1009; El Monte, CA. 91731

Burroughs Corporation
P. O. Box 142
El Monte, CA. 91734

attn: Publications Department
Technical Information Organization, TIO — West

FOLD UP FIRST FOLD UP

Printed in U.S.A. January, 1976 1089794