Burroughs

Language Manual

B 1000 Systems Data Management System II (DMSII) Host Language Interface

Distribution Code SC

Priced Item
Printed in U.S.A.
October 1986

5024516

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

INTRODUCTION

This manual provides key concepts regarding the interface between a host language and the B 1000 Data Management System (DMS) system. Although it primarily describes the COBOL74 and COBOL procedures for interfacing to DMS, RPG conventions are also noted.

Manual Structure

Section 1: Introduction
Section 1 includes a list of the components that form the nucleus of the DMS system, gives information on the three B 1000 DMS manuals, and includes a list of other B 1000 manuals that are pertinent to the B 1000 Data Management System.

Section 2: Host Language Interface
Section 2 provides general information about the interfaces between DMS and host languages. Included are discussions of the COBOL DATA DIVISION and the COBOL PROCEDURE DIVISION, programming notes pertinent to the COBOL compiler, and information on current record pointers.

Section 3: ANSI 74 COBOL Language Statements
Section 3 describes all the COBOL verbs used to manipulate data sets. COBOL compilation procedures are also included in this section.

Section 4: Audit and Recovery Restart Procedure
Section 4 provides information on restart procedures invoked during recovery processes.

Appendix A: Glossary of Terms
Appendix A contains a glossary of the DMS terms used in this manual.

Appendix B: COBOL Qualification of DMS Identifiers
Appendix B gives examples of COBOL qualification of DMS identifiers.

Appendix C: DMS Operation Summary
Appendix C provides summaries of the DMS verbs for RPG, COBOL, and COBOL74, and shows the differences between COBOL and RPG implementations of the DMS interface.

Appendix D: Notation Conventions and Syntax Specifications
Appendix D contains the standard explanation of notation conventions and syntax specifications.

Data Management System Components

The following components form the nucleus of DMS:

- A DMS Data and Structure Definition Language (DMS/DASDL) that describes a DMS data base.
- An ANSI 68 COBOL, ANSI 74 COBOL, or RPG language interface that provides programmatic access to the data in the data base.
- The DMS access routines, contained within the program DMS/ACR, that control storage and retrieval.
- The DMS/REORGANIZE program that is used in conjunction with the DMS/DASDL compiler and redescribes portions of the data base.

- The DMS/RECOVERDB program that automatically restores the integrity of a data base that has been corrupted through a system failure.
- Security features to protect the operating system and the data bases.
- Utility programs to assist in debugging the Data Management System and DMS data bases.
- DMS/INQUIRY, a program product that allows ad hoc query of a DMS data base.

DMS DOCUMENTATION

The overall data management system for B 1000 systems is described in the three documents identified and outlined in the paragraphs that follow.

DMS/DASDL Language Manual

Full title: B 1000 Systems DMSII Data and Structure Definition Language (DMS/DASDL) Language Manual.

The main text includes an exposition of the DMS structure types, identification and descriptions of the components of a data base, information on remap data sets and logical data bases, and a description of DMS/DASDL compilation.

The appendixes provide examples of DMS/DASDL physical structures, a DMS/DASDL glossary, the DMS/DASDL compiler messages, an example of data base development, and another example that shows the use of many of the elements of the DMS/DASDL syntax.

DMS Functional Description Manual

Full title: B 1000 Systems Data Management System II (DMSII) Functional Description Manual.

The main text describes the update and reorganization processes, the audit and recovery system, and data base security. Separate sections describe each of the following programs:

DMS/DECOMPILER
Reconstructs the original DMS/DASDL source of an existing DMS data base.

DMS/DASDLANALY
Decodes the contents of the data structures within a DMS data base dictionary.

DMS/DBLOCK
Locks the data base dictionary to block updating until this program terminates, thus providing protection against unwanted updating.

DMS/AUDITANALY
Decodes a DMS audit file and prints the content of each audit record.

DMS/DBMAP
Checks the integrity of a data base and prints structure information from the data base dictionary, performs population summaries, and prints data base data.

The appendixes provide summaries of the functions of the DMS/DASDL generated code, and record descriptions for all the DMS data structures referenced in the main text.

DMS Host Language Interface Manual

Full title: B 1000 Systems Data Management System II (DMSII) Host Language Interface Language Manual.

The main text includes general information on interfaces between the Data Management System and the host language (specifically COBOL74, with summaries of COBOL, COBOL74B, and RPG), descriptions of all the COBOL74 and COBOL language statements (verbs), a discussion of COBOL74 and COBOL compilation procedures, and audit and recovery restart procedures as they relate to the host language interface.

The appendixes provide information on qualification of DMS identifiers and a summary of DMS operations.

RELATED DOCUMENTS

The following manuals include information pertinent to the B 1000 data management system.

- *B 1000 Systems System Software Operation Guide, Volume 1,* form number 5024508.
- *B 1000 Systems COBOL Language Manual,* form number 1057197.
- *B 1000 Systems COBOL74 Language Manual,* form number 1168622.
- *B 1000 Systems Data Management System II (DMSII) Inquiry Language Manual,* form number 1108875.
- *B 1000 Systems DMSII Data and Structure Definition Language (DMS/DASDL) Language Manual,* form number 1152089.
- *B 1000 Systems Data Management System II (DMSII) Functional Description Manual,* form number 5016470.
- *B 1000 Systems Report Program Generator (RPG) Language Manual,* form number 1152063.
- *B 1000 Series Generalized Message Control System (GEMCOS) User's Manual,* form number 1093499.

SECTION 2
HOST LANGUAGE INTERFACE

GENERAL

This section provides general information regarding the interface between a host language and the Data Management System (DMS), with specific reference to the COBOL74 language syntax. A summary of the differences between the COBOL and COBOL74 languages is also given. Detailed explanations of the RPG language interfaces are included in the B 1000 RPG Language Manual.

Because interfaces to the three COBOL compilers are so similar, henceforth in this manual, the word COBOL is used to identify procedures relevant to any COBOL compiler. The individual compilers are specifically referenced, when differences exist, as the COBOL68, COBOL74, or COBOL74B compilers.

INTERFACE DIFFERENCES

The interfaces to the COBOL68 and COBOL74 compilers are identical with two exceptions:

1. The COBOL74 language allows retrieval of records either containing a given key value or conforming to a general expression composed of key values (the general selection feature). The general selection feature is not supported in COBOL68.
2. The COBOL74 verb LOCK is identical to the COBOL68 verb MODIFY. All other verbs are the same in both COBOL68 and COBOL74.

The interfaces to the COBOL74 and COBOL74B compilers are identical with three exceptions:

1. COBOL74 and COBOL68 compilers obtain information regarding the data base from DMS library files. The COBOL74B compiler, however, obtains its information directly from the DMS data base dictionary. Because of this the compiling procedures differ. Refer to Section 3 for additional information.
2. The COBOL74B compiler makes the DMSTATUS(DMSTRUCTURE) and DMSTATUS(DMERRORTYPE) values available when a DMS exception has occurred. COBOL74 and COBOL68 do not have this feature.
3. The COBOL74B compiler allows the definition of a DMSUSE procedure. The COBOL74B and COBOL68 compilers do not include this feature.

There are two interfaces between the host language and the data base system; one is used during compilation and one is used during execution. The compilation interface provides syntax that allows an application program to use any or all portions of a data base through the use of the INVOKE statement. In the INVOKE process, DMS/DASDL-generated library files (or, in the case of the COBOL74B compilers, the DMS dictionary itself) supply the language compiler with a description of the program-selected portions of the data base. The language 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.

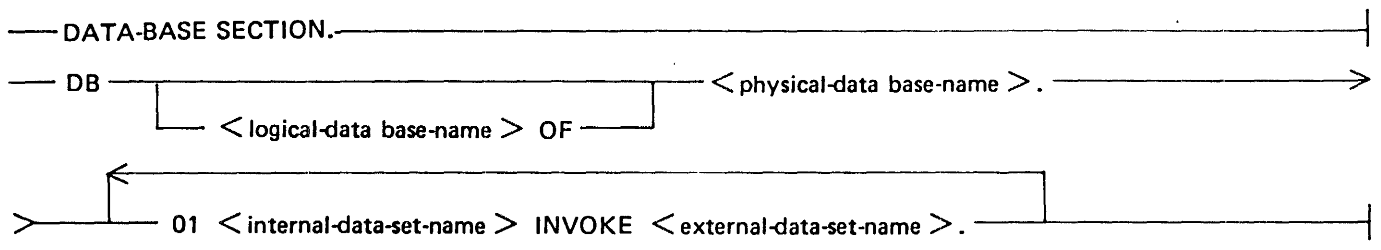
COBOL DATA DIVISION

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 of DATA DIVISION

In the DATA-BASE SECTION all disjoint data sets intended for use must be invoked. The compiler includes 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 and creates the interfaces to be used at run time.

Syntax:


DATA-BASE SECTION.
DB
< logical-data base-name > OF
< physical-data base-name > .
01 < internal-data-set-name > INVOKE < external-data-set-name > .

Semantics:

<DB>
The level indicator, DB, selects a particular data base. Only one data base can be selected in a program.

<logical-data-base-name>
The <logical-data-base-name> can be used as a qualifier of data set names. (The < physical-data-base-name> may be used if no < logical-data-base-name> is used.) The data-base-name is the family-name of the program-identifier used in the DMS/DASDL compilation (see Section 5 of the *B 1000 DMSII Data and Structure Definition Language (DMS/DASDL) Language Manual).* See Appendix B for a discussion of qualification of DMS identifiers in COBOL.

<logical-data base-name> must either be the name of the physical data base, or the name of a valid logical data base as described by a DATABASE statement in the DMS/DASDL source for <physical-data base-name>. When the physical data base is named, it is for documentation purposes only. COBOL68 and COBOL74 (but not COBOL74B) use the < logical-data-base-name> to locate the library files. The libraries are generated by the DMS/DASDL compiler when $COBOLIB cards are included in the source.

When using a logical data base, only data sets included in the list given in the DASDL source DATABASE declaration for that data base may be invoked as <internal-data-set-name>s.

<physical-data-base-name>
The <physical-data-base-name> is the name of the data base in the DASDL source file.

<internal-data-set-name>
The <internal-data-set-name> is the name of the data set in the COBOL source file.

<external-data-set-name>
The < external-data-set-name> is the name of the data set in the DASDL source file.

Example 1:

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

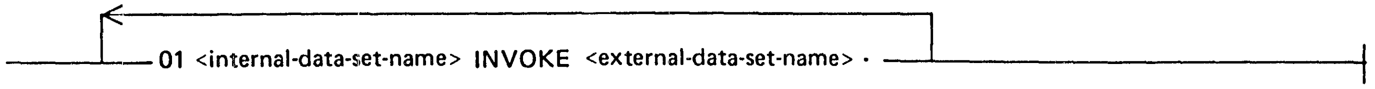
Example 2:

```
001031   DATA-BASE SECTION.
001032   DB   LDB1 OF UNIV.
```

Data Set References

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

Syntax:



Semantics:

<01>
The level number 01 selects particular disjoint data sets from a data base.

Each compilation copies the description of each invoked data set into the program from a library file created by the DMS/DASDL compiler (or, for COBOL74B, from the data base dictionary). The file-identifier of this library file for each data set has the following format:

#<data-base-name>/< data-set-name>

<internal-data-set-name>
The <internal-data-set-name> allows synonym capability and also allows multiple invokes of the same external data set, establishing more than one record area and current record pointer for that data set.

If a disjoint data set is invoked more than once in a program, each invocation of that data set must be assigned a unique < internal-data-set-name>. These unique names are necessary to allow proper qualification of data items within the data set.

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

Only disjoint data sets which are actually used within the program need be invoked. Disjoint data sets referenced by manual subsets but not explicitly used by the program need not be invoked.

<external-data-set-name>
The < external-data-set-name> is the name of the data set in the DASDL source file.

Example:

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

Invoked Data Set

The COBOL compilers print the names of all the paths and data items and also show the structure number, remap number, and version assigned during the DMS/DASDL compilation. The source statements supplied by the DMS/DASDL compiler are distinguished from the program source statements by an asterisk (*) character appearing to the left of the print line, as the coding example below indicates.

Example:

```
001940 /
001945   01 ADR INVOKE ADR.

*
*          01 ADR DATA SET (  9 12:59:20  3/ 8/77 ) .
*           SAD SET ( 19 12:59:20  3/ 8/77 , AUTO) OF ADR (  9 12:59:20
*             3/ 8/77 )
*               KEY IS ZIPC.
*          STUAD SET ( 21 12:59:20  3/ 8/77 , AUTO) OF ADR (  9 12:59:20
*             3/ 8/77 )
*               KEYS ARE ZIPC, SNO.
*          FACAD SET ( 22 12:59:20  3/ 8/77 , AUTO) OF ADR (  9 12:59:20
*             3/ 8/77 )
*               KEYS ARE ZIPC, SNO.
*          ADMAD SET ( 24 12:59:20  3/ 8/77 , AUTO) OF ADR· (  9 12:59:20
*             3/ 8/77 )
*               KEYS ARE ZIPC, SNO.
*             02 FACULTY-STUDENT                PIC 9 COMP.
*             02 SNO                            PIC 9(9) COMP.
*             02 NUMLNS                         PIC 9 COMP.
*             02 ADLN OCCURS 9 TIMES            PIC X(30).
*             02 ZIPC                           PIC 9(5) COMP.
*             02 PHON                           PIC 9(12) 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 reflect the usage of data items.

Multiply-Invoked Data Set

Since one record area can only hold one record at a time, it may be more effective 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 record pointers for the same set. Each current record 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.
```

Variable Format Records

The mechanism used by the COBOL and RPG compilers to process variable format records results in multiple redefinitions of the variable format parts of the data set record. In the COBOL compilers, this redefinition is accomplished explicitly through the use of the COBOL REDEFINES clause. Each variable format part is a group item, and each variable format part redefines the same physical area of the data set record. In the RPG compiler, the DMS/DASDL-generated library files contain offset and length information for each data item; an implicit redefinition of the variable format part of the data set record is easily accomplished by assigning common offsets to items which redefine the same data space.

When processing a data set that includes a variable format, no attempt is made, by the COBOL or RPG compilers, by any of the interpreters, or by the Data Management System, to ensure that only data items included in the current variable format part are manipulated or stored. The only checking that is performed on data items within the variable format part of a record is that specified by the programmer through the REQUIRED, VERIFY, and INITIALVALUE clauses. The programmer must ensure that items within a variable format part are only used by a program when the value of the RECORD TYPE field is equal to the value required for that item.

COBOL PROCEDURE DIVISION

Special extensions to the COBOL language 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 during selected data base operations.

MOVE and MOVE CORRESPONDING

The COBOL definition 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. The control information cannot be accessed in any way by COBOL programs. This includes the group MOVE operation, as the following example illustrates.

Example:

```
001930   01  MSF INVOKE MSF.
*
*        01 MSF DATA SET ( 13  9:12:54  3/15/77 ) .
*          MSFSET SET ( 18 9:12:54  3/15/77 , AUTO) OF MSF ( 13 9:12:54
*           3/15/77 )
*             KEY IS SSNO.
*          SSNSET SET ( 25 9:12:54  3/15/77 , AUTO) OF MSF ( 13 9:12:54
*           3/15/77 )
*             KEYS ARE SSNO.
*          02 SSNO                          PIC 9(9) COMP.
*          02 NONAM                         PIC 9 COMP.
*          02 LNAME                         PIC X(30).
*          02 QUARTER ORDERED DATA SET ( 15 9:12:54  3/15/77 )
*           QSET SET ( 15 9:12:54  3/15/77 )  OF QUARTER  ( 15  9:12:5
*          3/15/77 )
*             KEY IS QTR.                   PIC X(4).
*             03 QTR                        PIC 99 COMP.
*             03 QTTRHRS                    PIC 99 COMP.
*             03 QTRQP
```

The following is a functional description of the preceding example.

1. MSFSET, QUARTER, and QSET are control items not actually contained in the data set record. They are not moved in a MOVE MSF TO... or a MOVE...TO MSF operation.
2. QTR, QTTRHRS, and QTRQP are items of the record in the QUARTER data set and are not moved in a MOVE MSF TO ... or a MOVE...TO MSF operation.
3. The MSF record area for a group MOVE operation can be considered as the following items:

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

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

Exception Processing

The COBOL PROCEDURE DIVISION has been extended by the addition of DMS statements that provide an interface between a COBOL program and a data base. Any one of several exception conditions can be encountered during the execution of DMS statements that prevent the operation from being performed as specified.

If an exception condition occurs, the program terminates with a DS or DP condition unless the DMS statement is followed by an ON EXCEPTION phrase; or a USE ON DMERROR procedure has been declared (COBOL74B only). Therefore, if the program can continue to process in some fashion after an exception condition, an ON EXCEPTION phrase should be included. If the program terminates anyway when an exception is encountered, it is best to leave off the ON EXCEPTION phrase. This allows a dump to be taken.

Also, COBOL programs have access to a special register: DMSTATUS. The DMSTATUS register is set by the system at the completion of each DMS operation.

ON EXCEPTION Phrase

Each DMS statement yields a Boolean value which is TRUE if the operation resulted in an exception condition and FALSE if the operation completed with no exceptions encountered. If the Boolean value for the preceding statement is TRUE, <statement-1> of the ON EXCEPTION phrase is executed; otherwise, the next statement in the sequence is executed.

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

If an ON EXCEPTION phrase or a USE ON DMERROR procedure is not specified, the occurrence of an exception causes the program to be terminated with a DS or DP condition.

Syntax:

```
—— ON EXCEPTION < statement-1 > ——————————————————————————————|
```

The following example illustrates the ON EXCEPTION programming technique:

Example:

```
STORE COURSES
  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.
```

USE ON DMERROR Procedure

The COBOL74B compiler allows the declaration of a DMERROR procedure. This procedure is declared in the Declaratives Section, along with any other USE procedures. The USE ON DMERROR procedure is called each time an exception occurs during execution of a DMS function, unless an ON EXCEPTION clause is associated with that function. After execution of the procedure, control is returned to the statement following the DMS command. Refer to the COBOL74B appendix of the B 1000 COBOL74 Language Manual for a more complete discussion of USE procedures.

DMSTATUS Register

The DMSTATUS register is set by the system at the completion of each data management statement. When interrogating DMSTATUS, an <attribute name> in parentheses must follow the word DMSTATUS.

Syntax:

```
—— DMSTATUS ——┬— ( < attribute-name > ) —┬——————————————————————|
              ├— (DMCATEGORY) ———————————┤
              ├— (DMERROR) ——————————————┤
              ├— (DMSTRUCTURE) ——————————┤
              └—(DMERRORTYPE) ———————————┘
```

Semantics:

<attribute-name>
The <attribute-name> yields a TRUE value if the specified exception has occurred. These categories and their meanings are listed in this section.

DMCATEGORY
The DMCATEGORY attribute yields a numeric value identifying the exception. These values and their meanings are listed in this section.

DMERROR
The DMERROR attribute yields a TRUE value if any error has occurred.

DMSTRUCTURE
The DMSTRUCTURE attribute contains the number of the structure causing the exception. It is allowed only in COBOL74B programs. The structure numbers of all invoked structures are shown in the invocation information on the program listing.

DMERRORTYPE
The DMERRORTYPE attribute contains a numeric value identifying the subcategory of the exception. It is allowed only in COBOL74B programs. Subcategories are defined only for the DEADLOCK and DATAERROR exceptions. The value returned is zero for other exceptions. These values and their meanings are listed in this section.

The following is a list of each attribute-name and its description. Also listed is the DMCATEGORY value and the RPG indicator that can be specified for RPG/DMS programs.

DMERROR (D1 Indicator)
This attribute is set whenever any exception occurs. One of the following DMSTATUS exceptions is also set. Note for COBOL68 and COBOL74 programs that NOT DMSTATUS(DMERROR) is TRUE on a successful DMS operation. For RPG programs the D1 indicator is OFF on a successful DMS operation.

NOTFOUND (DA Indicator) 1
This attribute indicates that the specified record could not be found, locked, or modified.

FIND LAST or FIRST: The data set is empty.

FIND NEXT or PRIOR: There are no more records in the specified direction.

FIND AT KEY or FIND AT <expression>: No record meeting the conditions exists.

FIND NEXT AT KEY or FIND NEXT AT <expression> : No record meeting the condition is found following the current record.

FIND CURRENT: There is no current record or the current record has been deleted.

DUPLICATES (DB Indicator) 2
Duplicate keys not allowed in a set (STORE operation).

Duplicate keys not allowed in an ordered manual subset (INSERT operation).

Duplicate keys not allowed in an ordered embedded data set (STORE operation).

DEADLOCK (DC Indicator) 3
DMERRORTYPE 1: A deadly embrace occurred during an attempt to lock a record or do a BEGIN-TRANSACTION operation.

DMERRORTYPE 2: While attempting to lock a record, a program waited for more seconds than the value of the MAXWAIT parameter for its desired record to be unlocked (program status WAITING CONTENTION).

In either case, the Data Management System performs a FREE operation on all records locked by this program. The DMSTRUCTURE value returned is zero.

DATAERROR (DD Indicator) 4
DMERRORTYPE 1: An attempt was made to store a record with a null key, null required item, or fields that violated the DASDL-specified VERIFY condition.

DMERRORTYPE 2: An attempt was made to CREATE a variable format record with an invalid record type.

DMERRORTYPE 3: This value is not returned by B 1000 DMS. It has meaning in Burroughs A Series DMS.

DMERRORTYPE 4: An attempt was made to store a record with a changed value in a DASDL-specified READONLY item or in the variable format record type.

NOTLOCKED (DE Indicator) 5
This attribute indicates that a STORE operation was not preceded by a CREATE, RECREATE, or LOCK operation.

KEYCHANGED (DF Indicator) 6
This attribute indicates that when updating a disjoint data set record, a key used in an automatic set or subset with a NO DUPLICATES clause (the default) was changed or the key was changed when updating an ordered embedded data record.

SYSTEMERROR (DG Indicator) 7
This attribute indicates that there was a format error in a general selection expression. This attribute can also indicate that recovery is incompatible with the access routines (only returned to the DMS/RECOVERDB program).

READONLY (DH Indicator) 8
A STORE, DELETE, INSERT, or REMOVE operation was attempted on a data base that was opened inquiry only.

An attempt to change the value of a READONLY data item results in a DATAERROR (see DATAERROR) rather than a READONLY exception.

IOERROR (DJ Indicator) 9
An I/O error or corrupted index table control information was encountered when trying to read data from the data base. I/O errors encountered on a write operation are not returned to the program since write operations are not done explicitly by any program. An I/O error on a write operation sets a flag in the structure so that any subsequent access to it fails.

LIMITERROR (DK Indicator) 10
This attribute indicates that there is insufficient file space to add a record to a data set or to add a table to a list or set. When storing to a data set, the LIMITERROR might result if the data set or any of its automatic sets are full. In COBOL74B the DMSTRUCTURE attribute identifies the structure causing the LIMITERROR exception.

OPENERROR (DL Indicator) 11
This attribute indicates that a DMS operation has been attempted when the data base is not open, or an OPEN operation has been attempted when the data base is already open.

CLOSEERROR (DM Indicator) 12
This attribute indicates an attempt has been made to close a data base that is not open.

NORECORD (DN Indicator) 13
This attribute indicates that, when trying to access an embedded structure, no current record exists for the parent data set.

INUSE (DO Indicator) 14
This attribute indicates an attempt was made to delete a parent record for which embedded records still exist.

AUDITERROR (DP Indicator) 15
An attempt was made to do an update operation (STORE, DELETE, INSERT, or REMOVE verbs in COBOL68 and COBOL74 programs, and STORE, DELET, INSRT, REMOV operation codes in RPG programs) while not in transaction state.

An attempt was made to close a data base while in transaction state.

An attempt was made to perform a BEGIN-TRANSACTION (TRPEB in RPG) operation while already in transaction state.

An attempt was made to perform an END-TRANSACTION (TREND in RPG) operation when not in transaction state.

The AUDITERROR exception is only returned for audited data bases. If BEGIN-TRANSACTION or END-TRANSACTION operations are attempted for an unaudited data base, the program is terminated with DS or DP.

ABORT (DQ Indicator) 16
This attribute indicates that another program aborted or went to end of job (EOJ) while in transaction state. This error is returned to an update program when it closes the data base, does an END-TRANSACTION SYNC operation or a BEGIN-TRANSACTION operation. Refer to Section 4 of the *B 1000 Systems Data Management System (DMS) Functional Description Manual* for information on program aborts and their implications within other programs.

SECURITYERROR (DR Indicator) 17
This attribute indicates that an attempt was made to open a data base and the current usercode is not authorized. The SECURITYERROR exception is only returned for data bases that have a SECURITYGUARD file declared.

VERSIONERROR (DS Indicator) 18

This attribute indicates that an attempt has been made to open a data base that has had some structures redefined (with a $UPDATE and reorganization) since the program was compiled. The program needs to be recompiled.

FATALERROR (DT Indicator) 19

A fatal error is detected when either the access routines the data base is in serious trouble and cannot continue to process. Prior to the Mark 11.0 release, fatal error conditions halted the system. With the Mark 11.0 release, the use of access routines external to the MCP allowed fatal errors to be handled in a more user friendly way. Provided that memory corruption seems to be restricted to the data base itself, a DMS fatal error can be given rather than a system halt. In this case, the data base is closed for all programs and each receives a FATALERROR exception on their next DMS operation. Non-DMS jobs, or DMS jobs running on other data bases continue as normal. Application programs can be coded to detect a FATALERROR exception, re-open the data base, and go into recovery logic, without operator intervention.

From the point of view of the data base, a fatal error is treated like a Clear/Start operation; that is, no data buffers are written to disk, no disk file headers are updated and the data base requires Clear/Start recovery. The memory in use by DMS is properly returned to the system so that operation of the system as a whole is not affected.

The following actions occur when a FATALERROR exception is encountered:

1. The fatal error message is displayed on the ODT. The job detecting the fatal error is aborted (DS or DP).
2. Until the job is actually discontinued (DS or DP) other jobs can continue to run, unless they also detect a fatal error. This allows the operator to choose the best time for the fatal error.
3. When the job is discontinued (DS or DP) a system memory dump is automatically taken. This dump should be packaged and saved for analysis.
4. The data base is then closed. Any other job having the data base open receives a FATALERROR exception on its next DMS operation. Any other DMS operation after that (except for open) receives an OPEN exception, since at that time, no data base is open for the job.
5. The next data base OPEN operation initiates a Clear/Start recovery.

INTEGRITYERROR (DU Indicator) 20

This attribute indicates that there is corruption in the data base. When an integrity error first occurs on a structure, the access routines pause and a request is made for a system dump (DM system command) to be taken. The dump output should be packaged and saved for further analysis, along with the audit trails preceding the error. Following the dump, the program may be resumed with the OK system command. It then receives an INTEGRITYERROR exception, which it handles in whatever way has been programmed. Other programs are not affected by the integrity error if it does not impact their operation. Subsequent integrity errors on the same structure are returned to the program but do not cause the pause or request for a dump.

It is wise to bring the data base down shortly after an integrity error in order to examine it with the DMS/DBMAP program and repair it with a reorganization if necessary.

The following conditions may cause integrity errors:

- A key and data mismatch occurred on an automatic set or subset, but only if the KEYCOMPARE option is set.
- The automatic set or subset entry is missing during a DELETE operation.
- The automatic set or subset points to a deleted record.
- The program attempted to read an invalid logical address.
- A record on the "available" list is not marked as "dead."
- Addresses associated with the "available" list in the dictionary are bad.
- The Data Management System encountered invalid control information in an index table or list table.
- The Data Management System encountered missing areas in a data base file.

Selection Expression

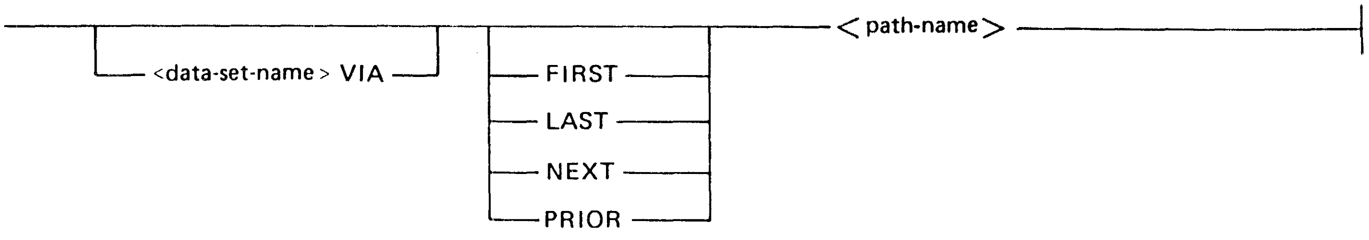
The selection expression specifies the desired record in a data set and also the record area into which the record is to be loaded. All record selections are made through paths. Paths are routes the system uses to locate records. The physical order of the records in a data set constitutes a path. Similarly, subsets, ordering keys, and retrieval keys are paths.

The verbs used with selection expressions are FIND and LOCK (FIND and MODIFY in COBOL68). Either of these verbs causes the record specified by the selection expression to be located and placed into the record area. If no record that satisfies the selection expression is found, a NOT FOUND exception is returned to the application program that made the request.

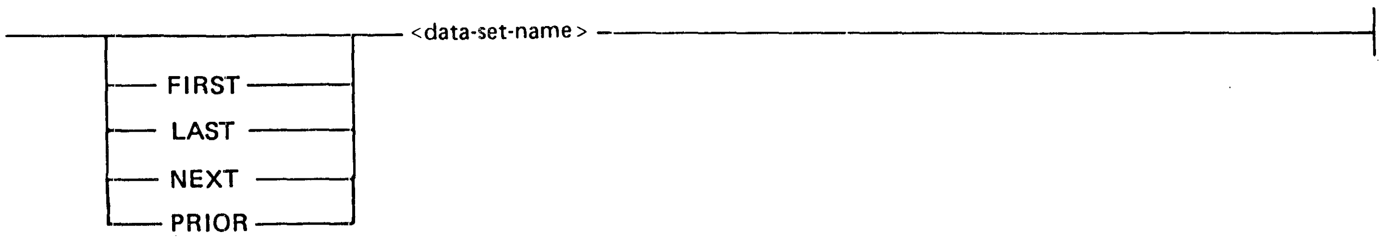
For a LOCK operation, the found record is locked so that a concurrent program cannot LOCK (MODIFY in COBOL68) or DELETE the same record. The current record pointer for the data set is updated and, if a set or subset is used, the current record pointer for the path is updated. Unused paths are unaffected. If an exception condition occurs, the current record pointers are not affected unless the exception returned was a NOTFOUND exception on a FIND AT KEY operation for an index sequential structure. Refer to Current Record Pointer in this section.

Syntax:

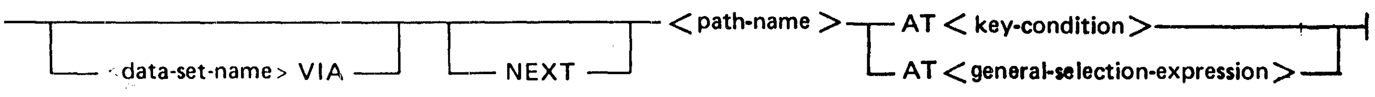
Form 1:



Form 2:



Form 3:



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

The optional phrase <data-set-name> VIA at the beginning of some forms of the selection expression is required when the path used is a manual subset and may also be required to qualify a multiply-invoked data set.

The phrase <data-set-name> VIA identifies the affected record area and current record pointer, provided that the desired record is found. By default, the data set is the data set referred to by the path used.

Examples:

```
FIND MSF VIA FIRST MSFSET
FIND FIRST MSF
LOCK MSFSET AT SSNO = C-SSNO
FIND MSF VIA MSFSET AT SSNO = C-SSNO
```

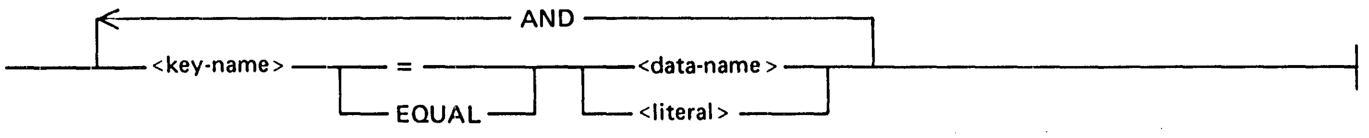
Path Name

The path name retrieves a record by way of the path. The path may be an automatic set or subset, a manual subset, or an access declared for an embedded ordered data set.

Key Condition

The key condition locates specific records in a data set spanned by a set or referenced by a subset. A key condition can be used by any COBOL compiler.

Syntax:



Semantics:

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

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

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

The key-names of a multi-item key must appear in the same order as specified in the DMS/DASDL source file.

Example:

```
FIND S AT A = 50 AND B = 50
```

General Selection Expression

COBOL68 programs cannot use general selection expressions. The general selection expression is a more general form of the key condition. The expression may be more general than that given for a < key-condition> -- any relational operator can be used, not all keys need be mentioned, any logical connective may be used. Refer to the definition of Condition in the B 1000 COBOL74 Language Manual. Refer to the heading entitled Generalized Selection Expression in this section for performance considerations.

Selection Expression Forms

The three forms of selection expressions are considered separately for discussion purposes.

Whenever an ordering is required but no explicit ordering exists, an implicit physical ordering is used. Whenever a current record pointer is required but is not in the proper state, the operation terminates with an exception.

Form 1

If no keyword is specified then the current record is found. This is the only mode of Form 1 in which the path may be an index random (retrieval) set.

The keyword FIRST specifies that the first record accessible by way of the specified path is to be selected. The path must not be an index random set. If a manual subset is used, the < data-set-name> VIA clause must be used. The record returned is the first in the physical order of the manual subset. If the manual subset is ordered, the entries are physically stored in key order so the physical and logical ordering are the same. Thus, a FIND FIRST finds the record with the lowest (or highest, for descending keys) key value.

The keyword NEXT specifies the next record to find by the path specified. The path must not be an index random set. Specifying this keyword for an ordered set or subset returns the record with the next higher (lower, if descending) key value. For ordered sets and automatic subsets, NEXT defaults to FIRST if this is the initial access of that set or subset since the data base was opened. For manual subsets and embedded data sets, NEXT defaults to FIRST if this is the first access since the current parent record was established.

The keyword LAST locates the last record in the specified path. The path must not be an index random set.

The keyword PRIOR locates the preceding record. The path must not be an index random set. NEXT and PRIOR are always relative to the current record pointer of the set. FIND PRIOR of a data set (Form 2) can return a different record than FIND PRIOR of a set (Form 1). The current record pointer of the set is updated to reflect the record located.

Example:

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

Since ascending sequence is the default ordering sequence for keys, the path K in the following example refers to members of D in sequence on A. A FIRST K therefore transfers 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 presented by K. If another ordering key, K1, was added with the specification K1 SET OF D KEY (A DESCENDING), the statement FIND FIRST K1 returns the member of D with the highest value of A.

Example:

```
DS DATA SET
    (A NUMBER (3);
     B NUMBER (10));
    K SET OF D KEY (A);
DS1 DATA SET
    (X NUMBER (4);
     Y SUBSET OF D;
     Z SUBSET OF D KEY (B);
     Z1 ALPHA (2));
```

If DS and DS1 are both invoked, the statement FIND DS VIA FIRST Y can then be used, returning the first physical record of DS in the table of subset Y for the current record of DS1. If the statement FIND DS VIA FIRST Z is used, the record found is that record of DS having the lowest value of B which was inserted into Z for the current record of DS1.

Form 2

If no keyword is specified, then the current record is found.

The FIRST keyword specifies that the record selected is the first physically located record in the file in which the data set is stored.

The NEXT keyword locates the record that physically follows the current record.

The LAST keyword locates the last physical record in the data set.

The PRIOR keyword locates the record that physically precedes the current record. The PRIOR keyword is valid only if the data set has already been accessed; otherwise, an exception condition is returned.

Form 3

If the NEXT keyword is omitted, the first record that satisfies the key condition is returned. If no record satisfies the key condition, a NOT FOUND exception is returned.

If the NEXT keyword is included, the access routines search forward from the current record pointer of the path, seeking a record to satisfy the condition. If the desired key follows the current record pointer, this is an implicit FIND FIRST operation. If the current record pointer follows the desired key, a NOTFOUND exception is returned.

The NEXT keyword in Form 3 finds all records with duplicate keys.

The path name in Form 3 may not be an unordered manual subset.

Example:

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

In the previous example, records of D could be selected based on the value of A using K, 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 = 100 AND B = 1001007890
FIND K1 AT C = B1
FIND D VIA K AT A = A1
```

Generalized Selection Expression

The generalized selection expression feature may be used by COBOL74, COBOL74B, and RPG DMS programs as well as the DMS/INQUIRY program. This feature allows application programs to select records based upon incomplete key information, and is explicitly limited to these four software products.

Although COBOL68 lacks this feature, it can achieve similar results programmatically. Refer to COBOL68 Programming Notes in this section.

For RPG programs, the DMKEY option to the FIND and LOCK operation codes in the Calculation Specifications is used for this function. This is done by specifying successive DMKEY options connected by AND or OR lines according to the normal RPG rules for the Calculation Specifications. Refer to the B 1000 Systems Report Program Generator (RPG) Language Manual for the complete syntax of DMS selection expressions in the RPG language.

No special action is needed to invoke the generalized selection expression. Whenever any selection expression is encountered by the COBOL74, COBOL74B, or RPG compilers, or the DMS/INQUIRY program, the Data Management System determines whether a simple DMS FIND/MODIFY operation satisfies the request or whether the expression qualifies as a generalized selection expression. If the expression is a generalized selection expression and the structure being used for the request is an index sequential set or subset, further analysis is performed to determine whether lower and/or upper bounds can be established for each field in the key. The Data Management System automatically positions to the lower bound for a FIND NEXT < generalized-selection-expression> operation if the current record pointer is less than the lower bound. If the current record pointer is greater than the upper bound, a FIND NEXT < generalized-selection-expression> operation returns a NOTFOUND exception condition.

The following example demonstrates the use of the generalized selection expression, using the CUST-SET set which references a CUSTOMERS data set. The example includes COBOL74 code which can be used to find all members of the CUST-SET set for which the CUSTOMER-TYPE key item (an alphanumeric item) is equal to the letter D.

Example:

DMS/DASDL Source:

```
CUSTOMERS DATA SET(
    CUSTOMER-TYPE      ALPHA(1);
    CUSTOMER-NUMBER    NUMBER(8);
       .
       .
       .
    );
CUST-SET SET OF CUSTOMERS KEY(CUSTOMER-TYPE,CUSTOMER-NUMBER);
```

COBOL74 Source:

```
    FIND CUST-SET AT CUSTOMER-TYPE = "D" ON EXCEPTION...
FIND-LOOP.
    .
    .
    .
    FIND NEXT CUST-SET AT CUSTOMER-TYPE = "D"
         ON EXCEPTION ... .
```

For index sequential structures, when the Data Management System actually encounters the communicate, the search is in two stages: (1) a binary search of the index tables for the entry which satisfies the lower bound and (2) a linear search of the tables, starting at the lower bound and continuing until either an entry is found which satisfies the request, or the upper bound is reached. For index random sets and ordered lists, only the linear search can be performed. For ordered lists, this search processes only the list tables pointed to by the current parent data set record for the list. For index random sets, all of the index tables for the index must be scanned.

Since any linear search is performed by the Data Management System, it can be costly in terms of processor utilization, though no more so than the same operations performed by an application program. The cost is a function of the following two factors: (1) the complexity of the selection expression, and (2) the physical length of the index tables being scanned.

As the logic within the selection expression becomes more and more complex, the likelihood decreases that a lower bound can be determined for the key items. If no lower bound can be determined for an item, zeroes are supplied. A lower bound of zero for a high-order key item results in a linear search which starts at or near the beginning of the index.

With regard to the physical length of the index tables being scanned, since the access routines must perform the linear search on an entry-by-entry basis, more entries cause longer searches.

Optimum performance can be derived from the generalized selection expression by stipulating a key condition in FIND or LOCK operations for which most of the work can be accomplished by a binary search, with as little work as possible being done by a linear search. The rules given below enable the analysis of a given selection expression. The rules determine whether the expression qualifies as a generalized selection expression and, if it does, whether a lower bound exists for the expression.

Even a complicated general selection expression runs more quickly than an application program that must perform the same work by reading each record and making tests on the relevant fields. This is because there is less need for communication between the application program and the access routines and also because the access routines need not read each record but must only scan the tables of the set.

Selection Expression Rules

Study the rules that follow. If a frequently-used selection expression requires linear search of all the tables of a large set, it is recommended that another set or, possibly, an automatic subset be added to the data base.

NOTE

In the rules that follow, the selection expressions do not conform in syntax to any of the user languages but are, rather, general representations of the conditions that can be coded in those languages.

Binary Search Conditions

If the following conditions are true, the expression can be satisfied by a FIND operation, using a binary search within the table to locate the exact entry requested:

1. The path specified is either an index random set, or an index sequential set or subset.
2. Each key item appears in the expression once.
3. Simple equality is specified between each key item and the value supplied.
4. All subconditions are connected by the AND logical operator.

All other selection expressions are generalized selection expressions.

Linear Search Condition

If the specified path is an index random set and any binary search condition is not met, a complete linear search must be performed on the entire index.

If the path specified is an ordered list, a linear search of the tables is always performed, regardless of the form of the selection expression.

Index Sequential Sets and Subsets

The following rules apply only to index sequential sets and subsets.

1. If the key items specified in the expression do not appear more than once but at least one item does not appear at all and all other binary conditions are met, then a lower bound of zero is supplied for each of the unspecified key values. There is one lower and one upper bound for each key. Starting from the left and proceeding right (as keys are declared in the DMS/DASDL source), the first key that is missing either an upper or lower bound causes the start of a linear search.

Example:

DMS/DASDL Source:

```
INVENTORY DATA SET (
     WAREHOUSE    NUMBER(2) ;
     PART-NO      ALPHA(6) ;
     BIN-NUMBER   NUMBER(4) ;
     ...
     );
PART-SET SET OF INVENTORY
     KEY (WAREHOUSE, PART-NO, BIN-NUMBER)
```

Selection Expressions

1) FIND PART-SET AT WAREHOUSE = 1 AND PART-NO = "A-123"
2) FIND PART-SET AT WAREHOUSE = 3 AND BIN-NUMBER = 15
3) FIND PART-SET AT PART-NUMBER = "A-123" AND BIN-NUMBER = 7

All three expressions are generalized selection expressions. Expression 1 results in a brief linear search, since the only unspecified key item is the BIN-NUMBER field, which is the least significant field. The result of this search is the first part number that is numbered A-123 in WAREHOUSE 1, regardless of bin number. Expression 2 results in a binary search that finds WAREHOUSE 3, then a linear search is made for BIN-NUMBER 15 on all paths in WAREHOUSE 3. Finally, selection expression 3 specifies only lower-order key items and leaves the major key item, WAREHOUSE, unspecified. The linear search for this expression includes the entire set.

2. If, instead of simple equality, the relational operators GREATER THAN (>) or GREATER THAN OR EQUAL (> =) appear in the expression, and all other conditions named by rule 1 remain true, then the lower bound generated for the expression is the same as in rule 1. Once that lower bound is found, the linear search may be, or need not be, more extensive.

   If any of the key items were declared to the DMS/DASDL compiler as DESCENDING, then the specification for that item can only contain the EQUAL (=), LESS THAN (<), or LESS THAN OR EQUAL (<=) operators for a non-zero lower bound to be established.

3. If the specification contains either of the LESS THAN or LESS THAN OR EQUAL operators for an ascending key item, or either of the GREATER THAN or GREATER THAN OR EQUAL operators for a descending key item, then the lower bound for any such item is zero. If all other conditions specified by rules 1 and 2 are true, then the linear search for such an expression is analogous to the types of linear searches described above with the exception that the LESS THAN operator establishes an upper bound, which, when met, serves to terminate the linear search at an earlier point than if no upper bound is specified.
4. If a NOT EQUAL (/=) operator is specified in a subcondition, zero is used as the lower bound for that key.
5. If, in addition to the conditions specified by rules 1 through 3, more than one subcondition is specified for a single key item, then both a lower bound and upper bound can be established for that item. Using the set described in rule 1, the following selection expressions illustrate this condition.

Selection Expressions:

```
1) FIND PART-SET AT
      WAREHOUSE = 4 AND PART-NO >= "B15" AND PART-NO < "C"

2) FIND PART-SET AT
      WAREHOUSE = 4 AND PART-NO >= "B15" AND PART-NO /= "B75"

3) FIND PART-SET AT
      WAREHOUSE = 4 AND PART-NO >= "B15" AND PART-NO < "C"
      AND PART-NO /= "B75"
```

All three expressions have the same lower bound. In addition, expressions 1 and 3 have the same upper bound. Expression 2 has an upper bound for WAREHOUSE only.

If no upper bound is specified for any item, there is always an implied upper bound with all bits on; that is, each digit contains the hexadecimal value @F@. Explicit upper bounds on other key items normally serve to terminate the linear search before any of the implied upper bounds are reached.

When using the generalized selection expression, follow the preceding rules to specify the desired expression. Ideally, the stated expression makes the most efficient use of the Data Management System and system resources. For example, the Data Management System returns the desired records if an expression which generates no lower bound is specified. However, the same results can often be obtained in less time, and with less impact on the rest of the system, by specifying several different expressions, each of which has a valid, non-zero, lower bound.

COBOL68 PROGRAMMING NOTES

This section is for those programmers using the COBOL68 compiler rather than the COBOL74 compiler. Although the methods given here may be used in COBOL74, the generalized selection feature in COBOL74 makes these methods unnecessary. The methods discussed here are not applicable for users of the RPG compiler.

A COBOL68 program that is using a path (set, subset, or access) to retrieve a data set record is required to specify all of the key items for that path. In addition, the only relationship that can be specified between the key items and the values supplied in a selection expression is simple equality. Also, all subconditions have to be logically connected by the AND operator. A single operation cannot specify relationships such as LESS THAN, GREATER THAN, or NOT EQUAL, or compound relationships. The OR operator cannot appear in a selection expression, and it is not possible to specify values for just a subset of the items that comprise a key.

Because of these restrictions, an extension to index sequential (ordered sets and automatic subsets) is available in the Data Management System. This extension, called the Partial Key Search, stipulates that the current record pointers for an index sequential structure be updated after FIND AT KEY and FIND NEXT AT KEY operations only, whether successful or not. The current record pointers for all other structure types, data sets, index random sets, and lists are updated only after a successful operation. Programs can perform a FIND NEXT operation on an index sequential structure, after a NOTFOUND DMSTATUS exception condition was returned on a FIND AT KEY operation on that same structure. The result is that the unsuccessful FIND AT KEY operation establishes a position within the index (at the point the key would have been), and the FIND NEXT operation can then retrieve records immediately after that position.

The following example demonstrates the use of the Partial Key Search technique, using the CUST-SET set which references the CUSTOMERS data set. The example includes COBOL68 source code, which finds all the members of the CUST-SET set for which the key item CUSTOMER-TYPE (an alphanumeric item) is equal to the letter D.

Example:

DMS/DASDL Source:

```
CUSTOMERS DATA SET(
    CUSTOMER-TYPE      ALPHA(1);
    CUSTOMER-NUMBER    NUMBER(8);
      .
      .
      .
    );
CUST-SET SET OF CUSTOMERS KEY(CUSTOMER-TYPE,CUSTOMER-NUMBER);
```

COBOL68 Source:

```
     FIND CUST-SET AT CUSTOMER-TYPE = "D" AND CUSTOMER-NUMBER = 0
          ON EXCEPTION IF DMSTATUS(NOTFOUND) FIND NEXT CUST-SET
          ELSE ...
FIND-LOOP.
     .
     .
     .
     FIND NEXT CUST-SET ON EXCEPTION ...
     IF CUSTOMER-TYPE = "D" GO TO FIND-LOOP.
```

Compare the above with the following B 1000 COBOL74 source.

COBOL74 Source:

```
FIND CUST-SET AT CUSTOMER-TYPE = "D"
     ON EXCEPTION ...
FIND-LOOP.
    .
    .
    .
    FIND NEXT CUST-SET AT CUSTOMER-TYPE = "D"
         ON EXCEPTION ...
    GO TO FIND-LOOP.
```

In the preceding example, the desired value is specified for the CUSTOMER-TYPE field, and zero is specified for the CUSTOMER-NUMBER field. By doing so, the programmer guarantees that the FIND AT KEY operation leaves the index pointing either at, or just before, the entry within the index that has the lowest possible value for the CUSTOMER-NUMBER field, as well as CUSTOMER-TYPE = D.

In general, partial keys can be used if a COBOL68 application program specifies values for the higher-order items within the key. The lower-order items have to be specified as low-values (zeroes for numeric items, @00@ for alpha). The limitations of the partial key are described as follows:

1. Partial keys can be used only for index sequential sets. No such capabilities exist for either index random sets or ordered lists.
2. No conditions can be described that ignore the high-order key items but give specific values for the low-order key items.
3. No compound expressions can be easily described. The following example of a generalized selection expression requires two different program loops to find the same record with the Partial Key Search technique:

```
FIND CUST-SET AT
     (CUSTOMER-TYPE = "D" AND CUSTOMER-NUMBER < 1500 ) OR
     (CUSTOMER-TYPE = "E" AND CUSTOMER-NUMBER < 300).
```

For these reasons, the generalized selection expression features available with the COBOL74 and RPG compilers are recommended.

CURRENT RECORD POINTER

Internally, the Data Management System maintains a current record pointer for each path (that is, for every set, subset, and for the data set itself) to each data set. The current record pointer points to a data set record and gives the Data Management System a reference point from which to move on subsequent FIND operations (for example, NEXT, PRIOR, NEXT AT KEY). Each invocation of a structure, in addition to providing another record area in the program, provides another current record pointer.

Here are the two typical cases:

- A FIND (or LOCK or MODIFY) operation of a data set by way of a path (including the data set itself) causes the current record pointer for that path to be changed to point to the newly found record, and also causes the current record pointer for the data set itself to be changed. Additionally, the data in the newly found record is put into the record area in the program.
- A STORE operation of a new record into a data set causes the current record pointer for the data set to be changed to point to the newly stored record, but does not update any current record pointers for sets or subsets.

Exceptions to these typical cases can best be understood if some background is gained concerning the internal workings of current record pointers.

Internal States

Internally, in the DMS/ACR (access routines) program, a current record pointer may be in one of three states: undefined, defined, or deleted.

Undefined State

The undefined state is its initial condition, and the condition into which a current record pointer of an embedded structure is placed each time a new record is found or stored in its parent structure. If a FIND NEXT operation is performed when the current record pointer is in this state, an implicit FIND FIRST operation is performed.

Defined State

The defined state is the normal condition of a current record pointer. In this state, operations such as DELETE may be performed on the current record as well as operations such as FIND NEXT, FIND PRIOR, and the like.

Deleted State

The deleted state is the state into which a current record pointer is moved when the record to which it referred is deleted. (See Deleted Records, later in this section.) When in this state, any operation (for example, DELETE) on the current record returns a NOTFOUND exception, but FIND NEXT, FIND PRIOR, and similar operations are still valid.

Create and Lock Flags

Each current record pointer for a data set has a create flag and a lock flag. The create flag is set if the last operation on the data set was CREATE or RECREATE. The lock flag is set if the last operation on the data set, by way of any path, was LOCK or MODIFY. Both flags are cleared by any other successful DMS operation on that data set, by way of any path, and are left unchanged by unsuccessful DMS operations.

Exceptions on FIND

If the FIND operation results in an exception, then the current record pointer for the path and the data set are not altered, with the following exception:

> The FIND operation was a FIND AT KEY operation, and the path was an index sequential set or subset, and the exception was a NOTFOUND.

In this one circumstance, the current record pointer for the path (but not the one for the data set) is updated to point to the position the record would occupy if it were there. This condition allows a subsequent FIND NEXT through that path to get the next key higher than the first specified value and enables the COBOL68 Partial Key Search operation. Refer to the partial key search information under the heading COBOL68 Programming Notes in this section. This exception applies only to the programs produced by the three COBOL compilers. It does not apply to RPG programs.

Deleted Records

When a record is deleted from a data set, any current record pointer (both path pointers and data set pointers) pointing to it in any program (including, of course, the one doing the DELETE operation) is changed to the deleted state.

Storing Records

In order to store a data set record, the current record for that data set must be either locked (lock flag is ON) state or created (create flag is ON) state. If it is in neither of these states, the STORE operation terminates with a NOTLOCKED exception. The state of the current record determines the kind of STORE that is done:

> If the create flag is ON, a new record is stored (create-store) and, if the STORE operation is successful, the current record pointer is moved to the newly stored record. That new record is locked and the create flag is turned OFF.

> If the lock flag is ON, the STORE operation updates the existing current record (update-store), the current record pointer is not moved, and the lock flag remains ON.

Unlocking Records

A data set record that is locked may not be locked (hence, may not be updated) nor deleted by any other program. The record is automatically unlocked when another DMS operation is done on the data set (discussed earlier). Records may also be explicitly unlocked with the FREE verb. In addition, all records except the restart data set record are unlocked at each END-TRANSACTION operation, and all records are unlocked when a DEADLOCK exception is reported. If a program abort occurs, all records for other programs are not only unlocked but are set to the undefined state.

After an abort occurs, the program may do additional LOCK operations before being notified of the ABORT exception on its next BEGIN-TRANSACTION operation. Therefore, it is possible for the program to have records locked at the time it first detects the ABORT exception.

Embedded Datasets

Current record pointers for embedded data sets follow the rules used for current record pointers of disjoint data sets, except that the state of the embedded data set current record pointer is dependent upon the state of the parent data set current record pointer. It is not possible to do any operation other than a CREATE operation on an embedded data set unless the current record pointer for its parent is in a defined state. Any such attempt results in a NORECORD exception.

Similarly, current record pointers for manual subsets follow the rules for current record pointers for automatic sets and subsets, except that the manual subset current record pointers are dependent upon the state of the parent data set current record pointer. Any attempt to use a manual subset when the current record pointer of the parent record is not defined results in a NORECORD exception.

Each time an operation capable of changing current record pointers is done on the parent data set, the current record pointers for all embedded structures are cleared, that is, the state is changed to undefined and the lock flag is cleared. The create flag is not cleared. Any FIND, LOCK, or MODIFY operation on the parent data set clears current record pointers for all embedded data sets even if the parent current record does not actually change (for example, FIND CURRENT).

SECTION 3
ANSI 74 COBOL LANGUAGE STATEMENTS

The following verbs are used by the COBOL compilers to manipulate data sets.

BEGIN-TRANSACTION
CREATE
DELETE
END-TRANSACTION
FIND
FREE
INSERT
LOCK (MODIFY in COBOL68)
RECREATE
REMOVE
STORE

In addition, syntax is implemented for the OPEN verb and additional semantics are given for the CLOSE verb.

The COBOL verbs, in alphabetical order, are described next along with the exceptions they can generate. Refer to the descriptions of exceptions in the DMSTATUS Register portion of section 2 for explanations of the exceptions.

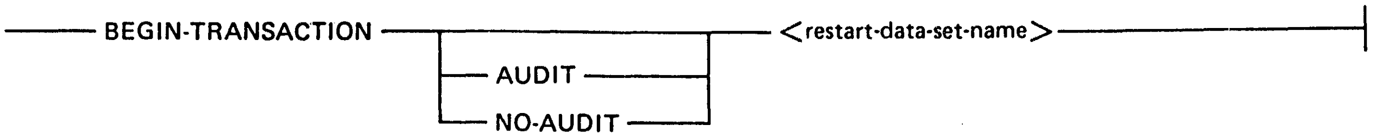
BEGIN-TRANSACTION

The BEGIN-TRANSACTION verb causes the program to enter transaction state. If an audited data base is in use, a program must be in transaction state before it performs any DMS operation that changes a record. Examples of such operations are STORE, DELETE, INSERT, and REMOVE. After the data base is opened or after the program has left transaction state by means of an END-TRANSACTION operation, the BEGIN-TRANSACTION verb must be specified before the first update operation.

If a program attempts a BEGIN-TRANSACTION operation while a Data Management System syncpoint or controlpoint operation or a program abort recovery is in process, the program is suspended at the BEGIN-TRANSACTION operation until the in-process operation completes.

The state of the program is unchanged if any exception condition occurs on a BEGIN-TRANSACTION operation. That is, if an AUDITERROR exception condition occurs because the program was already in transaction state, then the program remains in transaction state after encountering the exception. If any other exception occurs, transaction state was not entered; therefore, the program remains out of transaction state after encountering the exception.

Syntax:



Semantics:

AUDIT
If the AUDIT clause is specified, the Data Management System stores the current restart record of the program. This requires the user to have established a locked record in the restart data set by the time of the BEGIN-TRANSACTION operation. The programmer can lock a restart data set record by specifying the CREATE, RECREATE, or MODIFY verb.

The AUDIT clause is set TRUE by default.

NO-AUDIT
If the NO-AUDIT clause is specified, the STORE operation on the restart record is suppressed.

Exception Conditions:

ABORT
AUDITERROR
DEADLOCK
FATALERROR
INTEGRITYERROR
OPENERROR

If the AUDIT clause was specified, either implicitly or explicitly, any of the exception conditions which are possible on a STORE operation are also possible on a BEGIN-TRANSACTION operation.

CLOSE

The CLOSE verb closes a data base when further access is no longer required.

The CLOSE verb is optional because the system closes any open data base when the program terminates. However, an explicit close is recommended so that exception conditions can be returned.

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

If the data base is not open, the CLOSE operation terminates with a CLOSERROR exception condition.

If the program is currently in transaction state, an AUDITERROR exception is returned and the in-process transaction is rolled out by the DMS/RECOVERDB program, as though the program had been discontinued.

Syntax:

```
——— CLOSE <data-base-name> ————————————————————————————————|
```

Exception Conditions:

ABORT
AUDITERROR
CLOSEERROR
FATALERROR
IOERROR

CREATE

The CREATE verb causes space to be set up for adding a data set record. The CREATE verb must be performed prior to the addition of a new record in a data set (optionally the RECREATE verb can be used). A CREATE operation does not add the new record to the data base; that is the function of the STORE verb. The CREATE verb initializes the entire current record area of the data set according to the value of the INITIALVALUE clause in the data base description. Any data item in the data base description which does not have an INITIALVALUE clause is initialized to null (all bits ON). This initialization is used for validity checking of the record at the time of the STORE operation.

A CREATE operation causes an implicit FREE operation to be performed on the prior current record of the data set. The current record pointer goes to the created state.

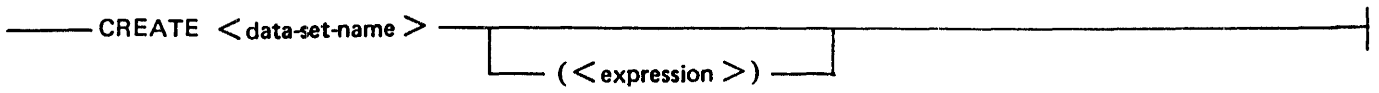
Normally, the CREATE operation is eventually followed by a STORE operation, which places the new record into the data set. However, if a subsequent STORE operation is not desired, the CREATE operation can be nullified by a subsequent FIND, MODIFY, CREATE, RECREATE, DELETE, FREE, or CLOSE operation. A BEGIN-TRANSACTION audit or END-TRANSACTION audit nullifies a CREATE operation on the restart data set.

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

A current parent record must exist if the data set is an embedded data set.

The CREATE operation is automatically performed for RPG programs that are adding records to a data set.

Syntax:



Semantics:

<expression>
<expression> is valid only if the data set description includes a variable-format part.

<expression> can be any of a literal, a simple data name, or an arithmetic expression. It cannot be a subscripted or indexed data name.

If the data set description referenced by the CREATE verb includes a variable-format part, an <expression> must be included in the CREATE verb of that data set.

If the value of <expression> does not match any of the values for the RECORD TYPE field declared to the DMS/DASDL compiler, then a DATAERROR exception condition is returned.

Exception Conditions:

DATAERROR
FATALERROR
OPENERROR

DELETE

The DELETE operation eliminates the current record from a data set.

A DELETE operation causes the current record area to be reloaded with the contents of the record.

If the record contains a non-empty embedded structure, an INUSE exception is returned. The record is not deleted.

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 programmer must remove the record from any manual subset that points to the data set record being deleted before deleting the record. (Refer to the REMOVE verb.)

Syntax:

```
—— DELETE <data-set-name> ——————————————————————————————————————|
```

Exception Conditions:

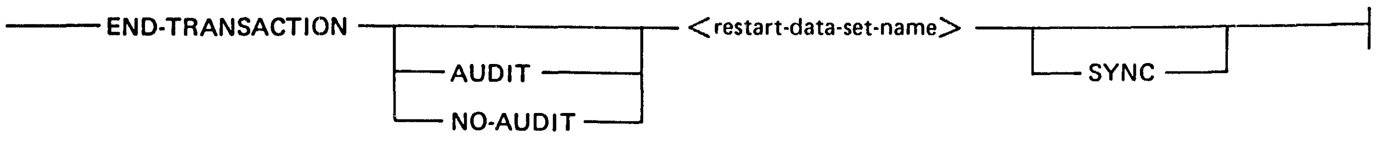
AUDITERROR
DEADLOCK
FATALERROR
INTEGRITYERROR
INUSE
IOERROR
READONLY
NORECORD
NOTFOUND
OPENERROR

END-TRANSACTION

The END-TRANSACTION operation causes the program to leave transaction state. This operation is normally executed at the end of any series of logically related updates to a data base. If a program is in transaction state immediately prior to closing the data base, an END-TRANSACTION operation must be performed before attempting to close the data base.

Except for an ABORT exception condition, the transaction state of the program is unchanged if an exception occurs on an END-TRANSACTION operation. If a program encounters an OPENERROR or AUDITERROR exception condition, the program remains out of transaction state. If any other exception condition occurs, the program remains in transaction state. If the program receives a FATALERROR exception, it is removed from transaction state and the data base is closed.

Syntax:



Semantics:

AUDIT
The AUDIT clause serves the same function as it does in the BEGIN-TRANSACTION verb.

NO-AUDIT
The NO-AUDIT clause serves the same function as it does in the BEGIN-TRANSACTION verb. The NO-AUDIT clause is set TRUE by default.

SYNC
The SYNC clause specifies that a syncpoint operation is performed after completing the END-TRANSACTION operation regardless of the actual number of transactions which have occurred since the last syncpoint operation. If a program attempts to perform an END-TRANSACTION operation with syncpoint after a program abort occurred, then the program is suspended at the END-TRANSACTION operation until the data base has been recovered. After the data base is recovered an ABORT exception condition is returned. When an ABORT exception condition is returned to a program that has attempted END-TRANSACTION operation with syncpoint, that last transaction of the program is backed out, and the syncpoint operation is not performed; however, the program is no longer in transaction state.

If the END-TRANSACTION operation is performed without a syncpoint and if another program aborts, then the last transaction of the program (and, possibly, previous ones) is backed out, but the program is not notified until the next BEGIN-TRANSACTION or CLOSE operation.

Exception Conditions:

ABORT
AUDITERROR
FATALERROR
OPENERROR

If the AUDIT clause was specified, any of the exception conditions that are possible on a STORE operation are also possible on an END-TRANSACTION operation.

FIND

The FIND operation performs two functions:

1. Locates the record satisfying < selection-expression>.
2. Transfers the data from the data base to the record area so it can be accessed by the program.

If the FIND operation results in an exception, then the current record pointer for the path and the data set are not altered, with the following exception:

> The FIND operation was a FIND AT KEY operation, and the path was an index sequential set or subset, and the exception was a NOTFOUND.

In this one circumstance, the current record pointer for the path (but not the one for the data set) is updated to point to the position the record would occupy if it were there. This condition allows a subsequent FIND NEXT operation through that path to get the next key higher than the first specified value and enables the COBOL68 Partial Key Search operation. Refer to the partial key search information under the heading COBOL68 Programming Notes in this section. This exception applies only to the three COBOL compilers. It does not apply to RPG.

If a record is found, the record is transferred to the record area of the program and the current record pointer is altered to refer to the found record. Also, if a set or automatic subset is involved, the current record pointer of the set or subset is altered to refer to the found record.

When performing a FIND operation by way of an embedded data set or manual subset, the current record pointer for the parent data set must be in a defined state.

Syntax:

```
—— FIND <selection-expression> ————————————————————————————————|
```

Exception Conditions:

FATALERROR
INTEGRITYERROR
IOERROR
NORECORD
NOTFOUND
OPENERROR
SYSTEMERROR

FREE

A FREE operation unlocks the current record.

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

A FREE operation is optional in most situations, since the CREATE, RECREATE, and sometimes the FIND or MODIFY operations perform an implicit FREE operation prior to their other actions. An implicit FREE operation is performed when any DMS operation that establishes a new current record pointer succeeds. Also, the END-TRANSACTION operation performs an implicit FREE operation on every record locked by this program, except for the restart data set.

The current record pointer and current record area are not affected by a FREE operation. If the current record pointer has just been created by way of the CREATE operation, it is changed back to its state before the previous CREATE operation; otherwise, it is not affected.

Syntax:

```
—— FREE <data-set-name> ————————————————————————————————————————|
```

Exception Conditions:

FATALERROR
OPENERROR

INSERT

The INSERT operation places a record into a manual subset.

The current record pointer of the data set must be defined. If it is not, the INSERT operation is terminated with a NOTFOUND exception condition.

The data set in which the manual subset is embedded must have the current record pointer in the defined state. If it is not, the operation is terminated with a NORECORD exception condition.

If duplicate keys are not allowed for an ordered manual subset and a record that has a key identical to that of the source record already exists in the manual subset, then a DUPLICATES exception condition occurs.

Syntax:

```
——— INSERT <data-set-name> INTO <manual-subset-name> ————————————————|
```

Semantics:

<data-set-name>
<data-set-name> must be the declared object of records for a manual subset.

<manual-subset-name>
<manual-subset-name> must be a manual subset of <data-set-name>, as the example below illustrates:

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

Exception Conditions:

AUDITERROR
DATAERROR
DEADLOCK
DUPLICATES
FATALERROR
INTEGRITYERROR
IOERROR
LIMITERROR
NORECORD
NOTFOUND
OPENERROR
READONLY

LOCK

The functions of the LOCK verb are identical to those of the FIND verb with one exception: if the record is found, it is locked, prohibiting other programs from performing a concurrent LOCK operation on the same record. However, other programs can perform FIND operations on locked records.

Since a record must be locked before it can be updated, a LOCK verb should be specified if there is a possibility that the data set record content is to be changed. The LOCK operation does not physically change the record but allows changes to be performed subsequently without a concurrent update from another program.

If the requested record is already locked by another program, a contention analysis is performed by the Data Management System. When performing a contention analysis, the Data Management System must examine the status of the program that has locked the requested record; if that program is waiting for another record to be unlocked, the Data Management System must repeat the process by examining the status of the program that has that next record locked. This process continues until one of the following occurs:

1. A program is encountered which is not waiting for a record to be unlocked. In this case, the program that was originally responsible for the contention analysis being performed is suspended by the Data Management System until either the record is unlocked or MAXWAIT seconds have elapsed. In the former case, the record is returned to the program; in the latter case, the program receives a DEADLOCK exception condition.
2. A program is encountered that is waiting for a record to be unlocked, and that record is locked by the program originally responsible for the contention analysis. This circular chain of programs is termed a Deadly Embrace. Upon recognizing this condition, the Data Management System returns a DEADLOCK exception condition to the lowest priority program in the chain and reinstates any programs that have been waiting for a record locked by that program.

Since no other program can lock a record once it is locked, it is important to free the record when it is no longer necessary to keep it locked. Unlocking a record is accomplished by a FREE operation or implicitly by a subsequent LOCK, FIND, CREATE, or RECREATE operation on the same data set. A subsequent STORE operation leaves the record locked. An END-TRANSACTION operation frees all records except the restart data set record.

Locking is maintained on a record level; other records within a block with a locked record are unaffected by the LOCK operation.

If the data set is embedded, or if the selection expression involves a manual subset, there must be a valid current parent record.

Syntax:

```
—— LOCK < selection-expression > ——————————————————————————————————————————|
```

NOTE

The LOCK verb is used in COBOL74 and RPG programs. The MODIFY verb is used in COBOL68 programs. The functions of LOCK and MODIFY are identical.

Exception Conditions:

DEADLOCK
FATALERROR
INTEGRITYERROR
IOERROR
NORECORD
NOTFOUND
OPENERROR
SYSTEMERROR

OPEN

The OPEN verb opens the data base.

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

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

The Data Management 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 following message is displayed:

NO FILE <data-base-name>/DICTIONARY

In addition, if the access routines (DMS/ACR files) are not present or are of an incompatible version, or if there is insufficient memory to open the data base, an appropriate message is displayed and the program hangs, waiting for an operator to correct the situation.

Syntax:

```
------ OPEN ---+--- UPDATE ----+--- <data-base-name> ------------------------------|
               |               |
               +--- INQUIRY ---+
```

Semantics:

INQUIRY
The INQUIRY clause specifies that any DMS operation (STORE, REMOVE, INSERT, and DELETE verbs) that changes the data base is not to be performed. Any such operation attempted on a data base opened with INQUIRY returns a READONLY exception.

UPDATE
The UPDATE clause specifies that DMS operations (STORE, REMOVE, INSERT, and DELETE verbs) that change the data base can be performed.

Exception Conditions:

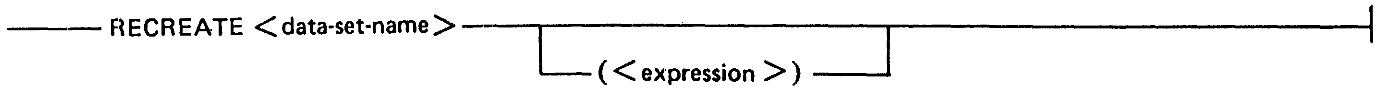
FATALERROR
OPENERROR
SECURITYERROR
SYSTEMERROR
VERSIONERROR

RECREATE

The RECREATE verb is identical to the CREATE verb with one exception: the record area for the data set is not completely initialized. All data items remain unaltered; however, items such as current record pointers for manual subsets and embedded data sets are set to the deleted state.

When a RECREATE operation is being performed, the Data Management System does not check to determine if the value of < expression> matches the contents of the RECORD TYPE field. If the values do not match, unpredictable results can occur when the program attempts to access data within the new current variable-format part after the RECREATE operation.

Syntax:



Semantics:

<expression>
<expression> can be a literal, a simple data name, or an arithmetic expression and is only valid if the data description includes a variable format part.

Exception Conditions:

DATAERROR
FATALERROR
OPENERROR

REMOVE

The REMOVE verb causes a record to be discarded from a manual subset. The manual subset must have a defined current record pointer. If it does not, the REMOVE operation is terminated with a NOTFOUND exception condition. The record referenced by the manual subset current record pointer is removed from the subset but not from the data set.

The data set in which the manual subset is embedded must have the current record pointer in the defined state. If it does not, the REMOVE operation is terminated with a NORECORD exception condition.

Syntax:

```
——— REMOVE  CURRENT  FROM <manual-subset-name> ———————————————————————————|
```

Exception Conditions:

AUDITERROR
DEADLOCK
FATALERROR
IOERROR
NORECORD
NOTFOUND
OPENERROR
READONLY

STORE

The STORE verb returns a modified record to a data set or places a newly created record into a data set.

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

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 and remains locked. If the current record pointer is in the defined state but unlocked or in an undefined state or deleted state, then the STORE operation terminates with a NOTLOCKED exception condition.

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

Current record pointers for sets are not affected by a STORE operation.

All fields that are, or form, part of a key or are specified with the REQUIRED clause in the DMS/DASDL source 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 STORE operation terminates with a DATAERROR exception condition.

Syntax:

```
——— STORE <data-set-name> ————————————————————————————————|
```

Pragmatics:

The following additional actions are performed depending on the prior DMS operation:

STORE Operation after CREATE or RECREATE Operation

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.

If an exception condition occurs that prevents the data record from being inserted into any of its sets, or into any automatic subset for which the WHERE condition is satisfied, then the STORE operation is terminated and the exception condition is returned to the program. In this case, the record is not inserted into the data set or into any of the sets or subsets. During the insertion of a key into a set or automatic subset, the DUPLICATES and LIMITERROR exception conditions can occur.

STORE Operation after LOCK Operation

If a STORE operation follows a LOCK operation, the record already exists in all sets.

If any terms involved in an automatic subset condition have changed, the condition must be reevaluated. The record is removed from the automatic subsets containing the record if the condition is not satisfied. The record is inserted into automatic subsets not already containing the record if the condition is satisfied.

If a key used in the ordering of a set is modified, and the record must be moved in that set, then the record is deleted from the set and reinserted in the proper position. A key must not be modified if duplicate records are not allowed or if the set is an embedded data set.

If a field used as a key field of a manual subset is changed, the STORE operation occurs, but no reordering of that manual subset is performed. It is the responsibility of the programmer to maintain manual subsets. On a FIND operation by way of a manual subset, if the object record key does not match the key in the manual subset, the object record is not found. The next record or, possibly, a NOTFOUND exception is returned.

Exception Conditions:

AUDITERROR
DATAERROR
DEADLOCK
DUPLICATES
FATALERROR
INTEGRITYERROR
IOERROR
KEYCHANGED
LIMITERROR
NORECORD
NOTLOCKED
OPENERROR
READONLY

COBOL COMPILATION PROCEDURES

Each COBOL compiler, when compiling a program containing DMS syntax, requires information about the structure of the data base. It uses this information to perform syntax checks, set up the program record areas, and to obtain internal identifications necessary to generate the code that communicates to the Data Management System at run time. The COBOL68 and COBOL74 compilers obtain this information from library files. The COBOL74B compiler obtains the information directly from the data base dictionary.

If the data base does not reside on system disk, then the compiler must be notified at compile time. For the COBOL68 and COBOL74 compilers this is done by equating the pack id of the internal LIBRARY file:

```
CO <myprog> COBOL74 LI;FI LIBRARY PID <dbpack>; ...
```

For the COBOL74B compiler this is done by equating the pack id of the internal DICTIONARY file:

```
CO <myprog> COBOL74B LI;FI DICTIONARY PID <dbpack>; ...
```

Library Files

Because the COBOL68 and COBOL74 compilers require library files, these must be present at the time of compilation. The library files are created at DASDL compile time if a $COBOLIB card is contained in the DASDL source. When logical data bases are used, a $COBOLIB <data base name> card must be included for each logical data base requiring COBOL libraries. The library files reside on the same disk pack as the data base dictionary and are named:

```
#<data base name>/<disjoint dataset name>
```

There is one library for each disjoint data set of each data base that contained a library request ($COBOLIB). Each library contains information about the disjoint data set and any embedded structures or sets it may have.

When DASDL $UPDATE compile is executed, new library files may be made. They also reside on the same disk pack as the directory and are named:

```
3<data base name>/<disjoint dataset name>
```

When the DMS/REORGANIZE program completes, it renames these files, replacing the "3" with a "#," so that subsequent COBOL compiles finds library files matching the current data base.

Version Timestamps

The libraries and the dictionary contain version timestamps for each structure. The version changes when the logical makeup of the structure is changed by a reorganization and is used to protect a program from viewing data according to an old format. The COBOL compilers include the version in the code file and, at data base open time, the version in the code file is checked against the version in the current dictionary. If the versions are different, then a change has been made to the format of the data base since the program was compiled. In this case, a VERSIONERROR exception is returned to indicate that the program needs to be recompiled. The program may also need to be recoded, depending on the nature of the changes. When compiling with the COBOL74 or COBOL68 compiler, it is important to be certain that the libraries resident on disk are the correct ones for the current data base. If the libraries are not the correct ones, spurious version errors may occur. With the COBOL74B compiler this problem cannot arise, as it gets its version timestamps directly from the current dictionary.

Recompiling for a Reorganization

It is possible to reduce the total time needed to do a reorganization and get the system running again. The only down time required is the time for the execution of the DMS/REORGANIZE program, as the data base may be up during the DASDL $UPDATE compile, and program recompilations. As soon as the reorganization is run, newly-compiled programs can begin execution immediately.

In order to take advantage of this method when using the COBOL68 or COBOL74 compilers, the library files generated by the DASDL $UPDATE compile must be manually changed to the correct name (see Library Files above). It is prudent to back up the current copy of the libraries first. To take advantage of this method when using the COBOL74B compiler, it is necessary to equate the internal DICTIONARY file to the temporary dictionary that resides on the data base pack and is named:

2<new data base name>/DICTIONARY

Normally, <new data base name> is the data base name. It differs only when the data base has been renamed in the DASDL $UPDATE compile.

SECTION 4
AUDIT AND RECOVERY RESTART PROCEDURES

The DMS audit and recovery system consists of:

1. Code, in the operating system and DMS/ACR files, that audits all data base updates to a disk or tape.
2. The DMS/RECOVERDB program that processes audit information to restore the integrity of a data base that has been compromised by an application programming failure, system error, or hardware malfunction.

Additionally, the audit and recovery system is designed to accomplish this task in much less time, and with much less programming or operational effort than recovery procedures written in a program. The audit and recovery system is described in the B 1000 Systems Data Management System II (DMSII) Functional Description Manual.

There are a few restart procedures that apply to all environments. One set, ordered or retrieval, must be declared for the restart set. For batch programs, the key field for this index might typically be the program-id. For data communication programs, the key might identify the station responsible for the input.

Since a STORE operation is performed on the restart data set at either the BEGIN-TRANSACTION or END-TRANSACTION operation, the program must update all of the relevant fields in its restart record immediately before executing that operation. There must be sufficient data stored within the restart record to enable a program to restart itself in each of the areas discussed in the paragraphs that follow.

INTERNAL PROCEDURES

Items which are required to maintain consistency and reproducibility of results, such as control totals or preprinted form numbers for checks or invoices, must be accessible through the restart record.

EXTERNAL PROCEDURES RELATED TO THE DMS SYSTEM

The program must be able to restore any critical record or path pointers to their state at the point of the recovery. This is usually more important for batch programs than data communication programs, since successive data communication transactions are typically unrelated, whereas batch programs may process sequentially through an entire structure.

EXTERNAL PROCEDURES NOT RELATED TO THE DMS SYSTEM

Input and output files and non-DMS managed files, such as COBOL74 ISAM files, must be present so that they can be repositioned.

GENERAL PROCEDURES

Interaction among the areas previously described must also be taken into consideration. For example, it may be necessary to reprocess the payroll for several employees, repeating the update operations to all of the relevant data sets within the data base and possibly creating or adding to other tape or disk files which are used by another application program; however, if paychecks were physically created prior to the failure, then the program, while in restart mode, must either not produce any new checks or automatically void any duplicate checks. The restart procedures for such an application program must allow for the entry of the last form number physically assigned. By comparing the numbers being assigned to the last number actually issued prior to the failure, the application program determines when to start issuing live checks.

NOTE

This example assumes a stand-alone mix. In designing the procedures required to restart such a program when several programs are updating the data base, the interaction among all programs must be considered in order to guarantee reproducibility of results.

RESTART RECORD HANDLING

Immediately after opening the data base, a program must locate and lock its restart record. If the operation is successful, the program can examine that record to determine if a recovery has occurred, and if so, the restart procedures can be executed. If the LOCK operation is unsuccessful, a CREATE or RECREATE operation must be performed at this time to establish a locked record for this program and prevent the program from getting a NOTLOCKED exception when it attempts its first BEGIN-TRANSACTION or END-TRANSACTION with audit operation.

Just before closing the data base, the following COBOL or COBOL74 code can be performed to delete the restart record of a program, assuming that the one restart record used by the program is still locked. (In the syntax, <exc> refers to exception-handling code.)

```
BEGIN-TRANSACTION NO-AUDIT < restart-data-set-name> <exc>.

DELETE <restart-data-set-name> < exc>.

END-TRANSACTION NO-AUDIT < restart-data-set-name> SYNC <exc> .
```

By deleting its restart record, the program insures that its restart procedures need to be executed after data base open only when the initial LOCK operation on the restart set is successful. Note that AUDIT is suppressed on both BEGIN-TRANSACTION and END-TRANSACTION operations since there is no need to restart this operation. Also, the specification of SYNC insures that, if another program aborts after this END-TRANSACTION operation but before this program can close the data base, the ABORT exception at the close can be ignored.

Another method that can be used, rather than the deletion of the restart record, is to maintain a batch number within the restart record. The current batch number is given to the program either from the ODT or an external file, and compared to the number within the restart record. If the two numbers match, this run is a continuation of an interrupted batch and the program must perform its restart routines. If the two numbers do not match, this is a new batch and no restart is necessary.

BATCH PROGRAMS

Batch programs are much easier to restart than data communications programs, since they usually deal with one easily retrievable input source, and it is relatively simple to maintain information concerning the position of that input file as well as any output or secondary input files.

A physical count of the number of input records processed can be used to reposition an input file. On restart, the record count is the number of records that are to be passed over. For ordered DMS structures or any other non-DMS ordered files, a key field accomplishes repositioning. However, output files other than disk cannot be physically repositioned. Therefore, multiple output card or tape files must be merged, and line printer files, especially in the case of preprinted forms, may require operator intervention.

A stand-alone batch program that can be readily restarted or rerun can eliminate overhead by using large transactions. Additionally, the syncpoint and controlpoint values may be raised temporarily with the SM system command. Batch programs which do not run in the stand-alone mode should not use large transactions because all jobs must wait for long transactions to complete when a syncpoint occurs.

DATA COMMUNICATIONS PROGRAMS

Restart procedures for on-line Data Management Systems are much more complicated to design and to code than those for batch systems. Not only is the input difficult to retrieve but many programs may interact in complex ways and each program may handle a number of users. No general rules can be given for on-line recovery because it is site and application dependent. The generalized message-control system (GEMCOS) provides recovery for on-line environments. (See the section titled Recovery in the B 1000 Series Generalized Message Control System (GEMCOS) User's Manual.)

BACKED OUT TRANSACTIONS

All forms of recovery result in some transactions being backed out. Although this loss is seldom critical in a batch environment, it is usually vital in an on-line environment. The syncpoint DMS/DASDL parameter controls the number of transactions that can be lost. Because no program can do a BEGIN-TRANSACTION if the syncpoint count has been reached with the syncpoint not yet done, setting the syncpoint parameter too low may cause programs to await syncpoint too frequently.

One guideline is to set syncpoint to the number of jobs which concurrently update the data base. The syncpoint parameters can be set with the SM system command as well as with a DMS/DASDL update.

The controlpoint parameter controls the amount of time a CLEAR/START recovery takes and is unrelated to the number of lost transactions.

APPENDIX A GLOSSARY OF TERMS

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

access
A method to reach a desired record of an ordered embedded data set.

access routines
The system program that supplies DMS services to application programs. It is normally named DMS/ACR.

application program
A user written COBOL or RPG program making use of DMS features.

automatic subset
An index sequential structure declared in the DASDL source with a WHERE clause. An automatic subset can be used to find a subset of data records chosen by some values they contain.

COBOL
In this manual, the term COBOL is used in a broad sense to include the COBOL68, COBOL74, and COBOL74B languages.

COBOL68
In this manual, the term COBOL68 is used to specify the B 1000 COBOL compiler.

COBOL74
In this manual, the term COBOL74 is used to specify the B 1000 COBOL 74 Compiler.

COBOL74B
In this manual, the term COBOL74B is used to specify the B 1000 COBOL 74B Compiler.

communicate
The method by which a user program requests system services. The compiler generates an appropriate communicate for each DMS verb.

contention
A condition in which a program is attempting to access a table entry or logical record within a physical block which has already been locked by another user. If the program waits on contention for more than MAXWAIT seconds, it receives a DEADLOCK exception. Refer to deadly embrace for additional information.

current record
The data record referenced by a current record pionter.

current record pointer
The Data Management System maintains a current record pointer for each path (that is, for every set, subset, and data set itself) to each data set. The current record pointer points to a data set record and gives the Data Management System a reference point from which to move on a subsequent FIND operation (for example, NEXT and PRIOR).

data set
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 embedded structures.

deadly embrace
A condition in which a chain of programs exists, each of which is waiting for contention to be resolved on a record while simultaneously having locked a record for which another program in the chain is waiting. Upon recognizing a deadly embrace, the Data Management System returns a DEADLOCK exception to the lowest priority program in the chain and unlocks all records locked by that program.

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. Standard data sets, sets, and automatic subsets are the only structures that are disjoint. Disjoint sets can only refer to disjoint data sets.

embedded
The condition of being dependent on a data set that is on a higher level; that is, the condition of a data set that is an item within a data set.

index
A table of pointers to a data set used to provide specified access to a data set.

index random sets
An index may be specified as index random in the DASDL source. Such a set may be used to retrieve records with specified key values but not in any particular order.

list tables
Manual subsets and embedded data sets are maintained in list form. Several entries may appear in each list table.

manual subset
A data set record can only be inserted into or removed from a manual subset by an application program. No automatic maintenance is done on manual subsets. Manual subsets may only be embedded structures and may only refer to disjoint data sets.

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 keys.

ordered lists
Ordered embedded structures, manual subsets and data sets, are maintained in ordered lists.

parent
A data set record that has dependent data sets is referred to as parent of the records of the dependent data set. A parent may itself be a record in an embedded data set. An embedded data set cannot be accessed without accessing the parent.

path
A structure to be used for retrieval of data set records.

path name
A path name specifies the path to be used to retrieve records. The path may be an automatic set or subset, a manual subset, or an access declared for a embedded ordered data set.

population
The number of records in a data set.

record
A record contains all the information that pertains to an entity.

set
A disjoint path to a data set, also called in index.

span
An index, whether ordered or retrieval, that references every record in a data set is said to span the data set. Subsets, whether automatic or manual, may span a data set, although typically they are not spanning sets.

subset
A path to some or all of the records of a data set. The criterion for membership in the subset can be specified to the DMS/DASDL compiler through a WHERE clause, in which case the subset is automatic and maintained through an index structure. Alternatively, records can be programmatically inserted into the subset, in which case it is a manual subset and is maintained by means of a list structure.

unordered
Not maintained in a user specified order.

APPENDIX B
COBOL QUALIFICATION OF DMS IDENTIFIERS

Unique identifiers are required in COBOL programs. If a data set is invoked more than once, separate internal names must be used so 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 source:

```
D1 DATA SET(
   A NUMBER (5);
   B NUMBER (3));
S1 SET OF D1 KEY (A), INDEX SEQUENTIAL;
```

Example:

COBOL source:

```
DB  DBASE.
01  D1 INVOKE D1.
01  DA INVOKE D1.

WORKING-STORAGE SECTION.
77 A PIC 99.        (Invalid because it cannot 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 C
DMS OPERATION SUMMARY

Descriptions of the DMS verbs and the differences between COBOL and RPG DMS are given in this appendix.

DMS VERB SUMMARY

In the following summary, DMS verbs are listed in alphabetic order along with a brief description. In most cases, the verb as presented applies to RPG as well as to the three COBOL compilers (COBOL68, COBOL74 and COBOL74B) which are grouped together here and referred to as COBOL. Where the forms differ, the alternate forms and other differences are noted.

BEGIN-TRANSACTION (COBOL)
TRBEG (RPG)
The BEGIN-TRANSACTION operation notifies the Data Management System that updates are to be performed on an audited data base. After execution of this operation, the program is in transaction state. All updates to an audited data base must be performed while the program is in transaction state. The BEGIN-TRANSACTION verb is used in COBOL programs; the TRBEG verb is used in RPG program.

CLOSE (COBOL only)
The CLOSE operation closes a data base. If no CLOSE operation is executed by a program, an implicit CLOSE operation is performed by the Data Management System when the program goes to end of job (EOJ). The RPG compiler generates a CLOSE operation for RPG programs at end of job.

CREATE (COBOL only)
The CREATE operation initializes all data items in a data set record to values specified in the DMS/DASDL source, or to nulls (all bits on). A STORE operation after a CREATE operation adds a new record to a data set. RPG programs perform an implicit CREATE operation prior to any STORE operation code if the letters ADD are specified in the Output-Format Specifications for the data set.

DELETE (COBOL)
DELET (RPG)
The DELETE operation discards a data set record. The record being deleted could have been located by either a FIND or a LOCK operation.

DMKEY (RPG only)
The DMKEY option of the FIND and LOCK operation codes in describes the selection criteria to be used to locate a record. The equivalent function in COBOL programs is performed by syntax contained in the FIND or LOCK operations.

END-TRANSACTION (COBOL)
TREND (RPG)
The END-TRANSACTION operation notifies the Data Management System that the current set of updates has completed. After execution of this operation, the program is no longer in transaction state. The data base cannot be closed by a program while in transaction state. The END-TRANSACTION verb is used in COBOL programs; TREND is used in RPG.

FIND
A FIND operation locates a record in a data set and the data in that record is transferred into the memory area of the program for processing. A FIND operation is implicitly performed in RPG programs for all primary and secondary disjoint data sets declared as input-only, indicated by the letter I coded in column 15 of the File Description Specifications for the data set.

FREE
The FREE operation unlocks a record. The FREE operation is implicitly performed whenever another FIND operation is performed on the same data set. A FREE operation is implicitly performed for all locked records except the restart data set whenever a program executes an END-TRANSACTION operation (COBOL) or TREND operation (RPG), or when a DEADLOCK exception is returned.

INSERT (COBOL)
INSRT (RPG)
The INSERT operation enters a data set record into a manual subset.

LOCK (COBOL74, COBOL74B, RPG)
MODIFY (COBOL68)
The LOCK (MODIFY) operation is the same as the FIND operation, but the requested record is to be locked in preparation for subsequent updating. No program can perform a LOCK (MODIFY) operation on a record that is already locked by another program; however, a FIND operation can be performed on a locked record. A record must be locked before it can be updated in the data base. A LOCK operation is implicitly performed in RPG programs for all primary and secondary disjoint data sets for which updating is allowed, indicated by the letter U coded in column 15 of the File Description Specifications for the data set.

MODIFY (COBOL68 only)
See LOCK.

OPEN (COBOL only)
The OPEN operation must be performed prior to any attempted access of a data base. No explicit OPEN operation code exists in RPG programs. The RPG compiler generates an OPEN operation for the RPG program at beginning of job.

RECREATE (COBOL only)
The RECREATE operation is similar to the CREATE operation. A subsequent STORE operation adds a record to a data set. The values of the items within the data set record are not initialized as in a CREATE operation; any value contained within an item prior to a RECREATE operation is maintained after the RECREATE operation. No form of this operation exists in RPG programs.

REMOVE (COBOL)
REMOV (RPG)
The REMOVE operation discards an entry from a manual subset. The entry that is removed is the one most recently found (FIND, LOCK or MODIFY via the manual subset).

STORE
The STORE operation adds or updates a record in the data base. The STORE operation updates an existing record when it is used after a LOCK (MODIFY) operation. The STORE operation adds a record by specifying the CREATE operation in COBOL programs, or STORE with ADD in RPG programs. A STORE operation is implicitly performed in RPG programs for disjoint data sets declared as output, indicated by the letter O in column 15 of the File Description Specifications for the data set.

TRBEG (RPG only)
See BEGIN-TRANSACTION.

TREND (RPG only)
See END-TRANSACTION.

DIFFERENCES BETWEEN COBOL AND RPG DMS

The differences in the implementations of the COBOL-DMS and RPG-DMS interfaces are described in the following paragraphs. Unless noted, COBOL refers to the COBOL68, COBOL74, and COBOL74B compilers.

Data Storage

COBOL All data items are maintained directly within the record area that the Data Management System uses as its source and destination for STORE and FIND operations. Changes made to an item within this record area are automatically reflected in the data base after a successful STORE operation.

RPG The programmer has no access to the record area used by the Data Management System. The data items referenced in the Input Specifications are filled from the record area by RPG after a FIND or LOCK operation code. On output, only those items explicitly coded in the Output-Format Specifications are moved back into the record area; thus, an item can be modified in memory, but the data base copy of that item need not be changed.

Data Types

COBOL All data types permitted in the DMS/DASDL syntax are valid in the COBOL language.

RPG The RPG compiler does not allow unsigned numeric data items. Any numeric item described to the DMS/DASDL compiler as unsigned has a positive sign appended when the item is moved out of the DMS record area; this sign is removed when the item is moved back into the record area immediately prior to a STORE operation. These operations are performed automatically at the time the data is moved into or out of the DMS record area, and these operations do not add appreciable overhead to the execution of the program. However, if the value of the item becomes negative during the normal execution of the RPG program, no error is detected at the time the sign is stripped away from the item. The programmer must ensure that RPG programs that process unsigned numeric data items are not allowed to change the signs of such items.

Subscripting

COBOL	A maximum of three levels of subscripts are allowed. Each subscript must be less than or equal to 1023 (this is a DMS/DASDL restriction, not a COBOL restriction).
RPG	Only one subscript is allowed. As in the case of COBOL subscripts, the maximum value of a subscript has a DMS/DASDL-enforced limit of 1023. The method for remapping a multiply-subscripted item as an RPG-compatible, singly-subscripted item is described in Section 4 of the B 1000 Systems DMS Data and Structure Definition Language (DMS/DASDL) Language Manual.

Group Items

COBOL	The subitems within a group constitute an implicit redefinition of the group item. Therefore, changes at the group level are reflected within the subitems of the group, and changes to the subitems are similarly reflected within the group item.
RPG	The concept of a group item does not exist in RPG programs. If a group item is referenced in an RPG program, and the individual items in that group are also referenced, then the group item and all of its subitems are maintained separately. Each of the items is filled separately from the DMS record area after a FIND or LOCK operation. Changes to the group item do not affect any of the subitems, and changes to the subitems do not affect the group item. When the record containing the group is stored back into its data set, the order in which the group item and its subitems are coded in the Output-Format Specifications determines the final contents of the data set record.
COBOL	Group items can be subscripted.
RPG	Group items that are subscripted cannot be used in an RPG program.
COBOL	Group items can be nested; that is, a subitem within a group can also be a group.
RPG	Nested group items cannot be used in an RPG program.

Selection Expressions

COBOL68	All key items must be included, and the only relationship that can be specified between a key item and the value supplied is simple equality. The partial-key search function is allowed only on index sequential structures (ordered sets and automatic subsets).
COBOL74B	The COBOL74 and COBOL74B compilers allow generalized selection expression. The partial-key search function is also available.
RPG	The generalized selection expression is allowed. The partial-key search function is not allowed for RPG programs.

Library Files

COBOL68 & COBOL74 — These compilers include COBOL source library files describing the data base. Refer to COBOL Compilation Procedures in Section 3 for more detail.

COBOL74B — This compiler obtains its data base information directly from the dictionary. The record description included in the listing is produced by the compiler from the information encoded in the dictionary. Refer to COBOL Compilation Procedures in Section 3 for more detail.

RPG — The RPG library files contain encoded information about each of the items in the disjoint data sets, including sets, subsets, embedded data sets, and data items. These library files are not added to the RPG program source as COBOL library files are; rather, the RPG compiler locates these files when a data set is named on a File Description Specification and uses the information contained therein when processing the Input, Calculation, and Output-Format Specifications. The listing of the COBOL-type record description, included by the RPG compiler at the end of the compile listing, is generated by the RPG compiler from the encoded information in the RPG library file.

DMS Verbs

COBOL — All DMS operations must be explicitly performed by the application program; the COBOL compiler does not generate any hidden code for the Data Management System.

RPG — The following operations are implicitly performed by code generated by the RPG compiler:

> Data base OPEN and CLOSE.
>
> All FIND/LOCK and STORE operations on cycle-driven data sets.
>
> Any CREATE operation required for storing an added record.
>
> TRBEG and TREND, if a noncycle-driven data set in an audited data base is being updated.

COBOL — When adding records to data sets, either the CREATE or RECREATE operation must be explicitly specified prior to the STORE operation.

RPG — An implicit CREATE operation is generated by the RPG compiler immediately prior to every STORE operation of an added record. No explicit CREATE operation code exists in RPG programs, nor does any RECREATE operation, whether explicit or implicit.

COBOL The audit operation is performed by default for the BEGIN-TRANSACTION operation; no audit operation is performed by default for the END-TRANSACTION operation. The programmer can override either of these defaults, as well as request a syncpoint with the END-TRANSACTION operation.

RPG The default for the TRBEG operation code is no audit, for both implicit and explicit executions of this operation. For explicit executions of the TREND operation code, the default is audit; this can be overridden by the programmer. A syncpoint can be requested with an explicit TREND operation code. For implicit executions of the END-TRANSACTION (TREND) operation, if the LR indicator is ON, then by default no audit and syncpoint are performed by the RPG compiler-generated code. If the LR indicator is OFF, audit is set by default. Only the defaults for the explicit TREND operation codes can be overridden by the programmer; the defaults for the implicit BEGIN-TRANSACTION (TRBEG) and END-TRANSACTION (TREND) operations cannot be overridden, nor can the defaults for the explicit execution of the TRBEG operation code.

APPENDIX D
NOTATION CONVENTIONS AND SYNTAX SPECIFICATIONS

NOTATION CONVENTIONS

The following paragraphs describe the notation conventions used in this manual.

Left and Right Broken Brackets (<>)

Left and right broken bracket characters are used to enclose letters and digits which are supplied by the user. The letters and digits can represent a variable, a number, a file name, or a command.

Example:

<job #>AX<command>

At Sign (@)

The at sign (@) character is used to enclose hexadecimal information.

Example:

@F3@ is the hexadecimal representation of the EBCDIC character 3.

The at sign (@) character is also used to enclose binary or hexadecimal information when the initial @ character is followed by a (1) or (4), respectively.

Examples:

@(1)11110011@ is the binary representation of the EBCDIC character 3.

@(4)F3@ is the hexadecimal representation of the EBCDIC character 3.

<identifier>

An identifier is a string of characters used to represent some entity, such as an item name composed of letters, digits, and hyphens. An identifier can vary in length from 1 to 17 characters. The characters must be adjacent, the first character of an identifier must be a letter, and the last character cannot be a hyphen.

<integer>

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

<hexadecimal-number>

A hexadecimal number is specified by a string of numeric digits and/or the characters A through F; this string is enclosed within the at sign (@) characters.

<delimiter>

A delimiter can 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. A literal can be either an alphanumeric (or simply alpha) literal, or a numeric literal. Alpha literals can contain any combination of valid printable characters, or spaces, and must be enclosed by quotation (") characters; a quotation character within an alpha literal is represented by two successive quotation characters within the character string.

Example:

ABC""DEF

The preceding alpha literal could be used to represent the character string ABC"DEF.

Numeric literals can contain only the decimal digits 0 through 9 and are not enclosed within any delimiters.

Percent Sign (DMS/DASDL Only)

The percent sign (%) character designates DMS/DASDL documentation or comments, and its presence terminates the scan of a source record. The example below illustrates the use of a percent sign for in-line coding.

Example:

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

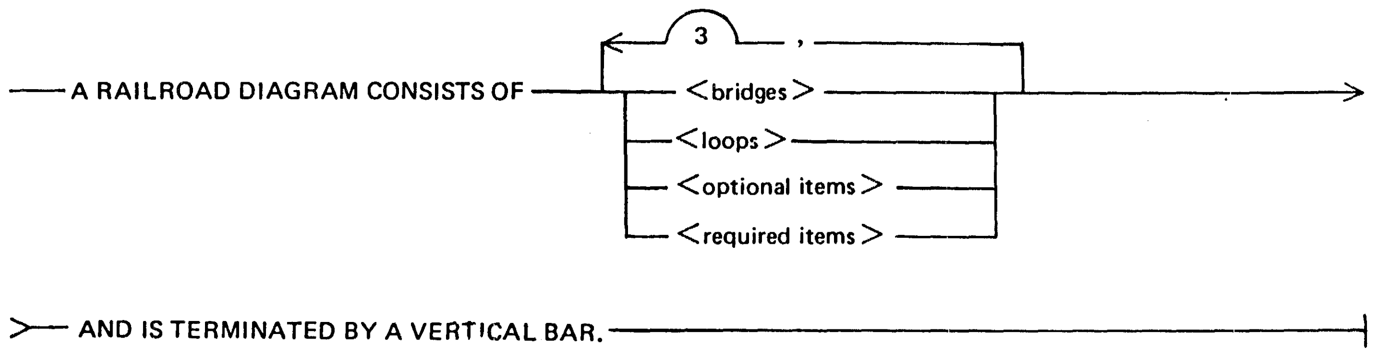
SYNTAX CONVENTIONS

Railroad diagrams show how syntactically valid statements can be constructed.

Traversing a railroad diagram from left to right, or in the direction of the arrowheads, and adhering to the limits illustrated by bridges produces a syntactically valid statement. Continuation from one line of a diagram to another is represented by a right arrow (→) appearing at the end of the current line and the beginning of the next line. The complete syntax diagram is terminated by a vertical bar (|).

Items contained in broken brackets (< >) are syntactic variables which are further defined or require the user to supply the requested information.

Upper-case items must appear literally. Minimum abbreviations of upper-case items are underlined.


A RAILROAD DIAGRAM CONSISTS OF
3
,
<bridges>
<loops>
<optional items>
<required items>
AND IS TERMINATED BY A VERTICAL BAR.

G50051

The following syntactically valid statements can be constructed from the preceding diagram:

A RAILROAD DIAGRAM CONSISTS OF <bridges> AND IS TERMINATED BY A VERTICAL BAR

A RAILROAD DIAGRAM CONSISTS OF <optional items> AND IS TERMINATED BY A VERTICAL BAR.

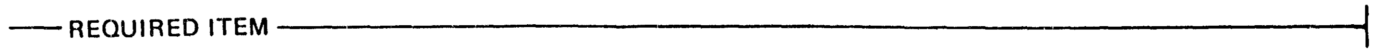
A RAILROAD DIAGRAM CONSISTS OF <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR.

A RAILROAD DIAGRAM CONSISTS OF <optional items>, <required items>, <bridges>, < loops> AND IS TERMINATED BY A VERTICAL BAR.

Required Items

No alternate path through the railroad diagram exists for required items or required punctuation.

Example:

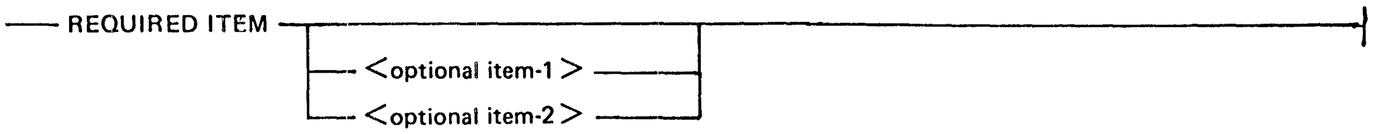


G50052

Optional Items

Items shown as a vertical list indicate that the user must make a choice of the items specified. An empty path through the list allows the optional item to be absent.

Example:



G50053

The following valid statements can be constructed from the preceding diagram:

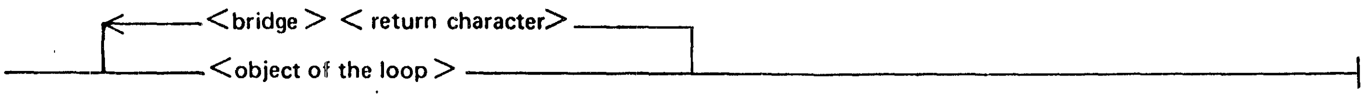
REQUIRED ITEM

REQUIRED ITEM <optional item-1>

REQUIRED ITEM <optional item-2>

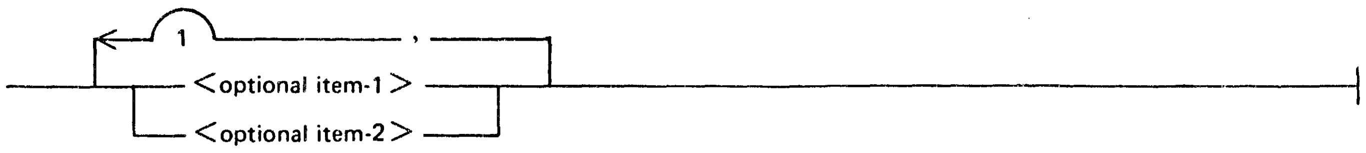
Loops

A loop is a recurrent path through a railroad diagram and has the following general format:



G50054

Example:



G50055

The following statements can be constructed from the railroad diagram in the example:

<optional item-1>

<optional item-2>

<optional item-1>,<optional item-1>

<optional item-1>,<optional item-2>

<optional item-2>,<optional item-1>

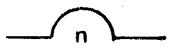
<optional item-2>,<optional item-2>

A <loop> must be traversed in the direction of the arrowheads, and the limits specified by bridges cannot be exceeded.

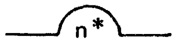
Bridges

A bridge indicates the minimum or maximum number of times a path can be traversed in a railroad diagram.

There are two forms of <bridges>.



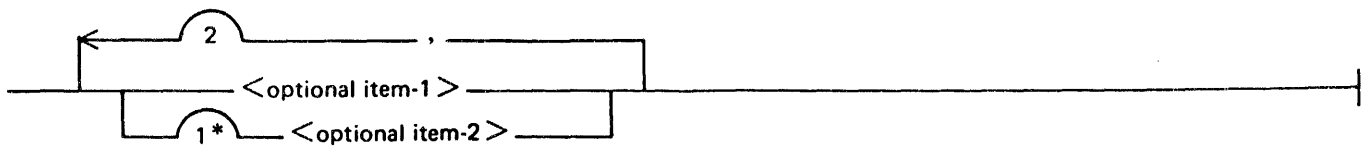
n is an integer which specifies the maximum number of times the path can be traversed.



n* is an integer which specifies the minimum number of times the path must be traversed.

G50056

Example:



G50057

The loop can be traversed a maximum of two times; however, the path for <optional item-2> must be traversed at least once.

The following statements can be constructed from the railroad diagram in the example:

<optional item-2>

<optional item-1>,<optional item-2>

<optional item-2>,<optional item-2>,<optional item-1>

<optional item-2>,<optional item-2>,<optional item-2>

INDEX

Documentation Evaluation Form

Title: B 1000 Systems Data Management System II (DMSII) Host Language Interface Language Manual

Form No: 5024516

Date: October 1986

Burroughs Corporation is interested in receiving your comments and suggestions, regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ **Addition** ☐ **Deletion** ☐ **Revision** ☐ **Error**

Comments:

From:

Name

Title

Company

Address

Phone Number

Date

Remove form and mail to:

Burroughs Corporation
Documentation Dept., TIO - West
1300 John Reed Court
City of Industry, CA 91745
U.S.A.