Burroughs Corporation

COMPUTER SYSTEMS GROUP
SMALL SYSTEMS PLANT

B1800/B1700 DMS/INQUIRY

P. S. 2222 2566

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	11/10/78	Original Release - Mark 8.0 Level		J Hale

TABLE OF CONTENTS

GENERAL

The DMS INQUIRY facility is a convenient method by which a terminal can be used to extract information from a database produced by the DMSII data management system.

The inquiry language enables nonprogrammers to utilize the system with little training, yet retains the ability to perform very complex functions. At the same time, the language contains constructs for compacting complex operations into simpler, single-identifier "macro calls". This flexibility, along with more powerful record selection capabilities, allows this facility to be useable in a wide variety of applications.

Section 1 contains the design objectives and a general description of the features.

Section 2 is intended as an introduction to the use of the system. The inquiry capabilities are presented by the extensive use of examples.

Section 3 contains the formal definition of the inquiry capabilities. In addition, it contains a formal definition of other features intended to aid and/or simplify a user's terminal interface to the system and the database.

RELATED PUBLICATIONS:

NAME	NUMBER
DADSL	P.S. 2219 0433

DESIGN OBJECTIVES

LANGUAGE INTERFACE

1. DMS/INQUIRY is an on-line interactive inquiry facility. While intended for programmer use, it is simple enough that with some training it could be easily used by nonprogrammers.

2. DMS/INQUIRY can extract information from any part of a database regardless of the complexity of the database.

3. DMS/INQUIRY takes advantage of any indexing structure, if possible, in extracting information from a data base.

4. DMS/INQUIRY always produces the requested information even if, to satisfy the request, it is necessary to do linear file searches. While this may take some time, an answer can be produced sooner than a user can design, program and debug a program that will satisfy the same request.

5. DMS/INQUIRY allows a terminal user to interrogate the description of a database.

6. DMS/INQUIRY has been kept as simple as possible, commensurate with previous objectives, by minimizing the number of statement types. This has been accomplished by defining basic functions which can be connected utilizing English-like connectors and/or qualifiers to form complex inquiry statements.

TERMINAL USER'S INTERFACE

1. The terminal attributes (i.e., page size, line width, etc.) are known to DMS/INQUIRY. This allows appropriate output formatting by DMS/INQUIRY without requiring the user to specify these at run time.

2. The language provides a facility for the user to display or alter terminal attributes.

GENERAL SYSTEM FEATURES

The DMS/INQUIRY facility assumes that the user has a basic familiarity with the structure of the data base. For example, he must know the names of the items he wishes to display or test; if embedded structures are employed, he must know the "nesting level" of the structures he references; he may need to know structure names for item qualification if the system is unable to determine from context which structures are required. The user may enhance his familiarity with these and other aspects of physical structure, if necessary, by means of certain inquiries which show the description of selected portions of the data base.

The most basic operation of DMS/INQUIRY is to select a record from the database and print values of certain items in that record. The user, by associating display lists and selection conditions with structures of the database, can control the manner in which records are selected and the frequency and amount of information printed at his terminal.

DMS/INQUIRY makes it easy to modify, extend or refine previously specified selection conditions or display lists. The user may subset structures into smaller collections of data with common properties, and restrict subsequent attention to the subsets.

The user may define "virtual items" which are functions of actual or other virtual items; virtual items may be used like actual items for selection or display purposes.

The data base administrators, through the DASDL remaps and logical data base facility, can protect against unauthorized access to sensitive data by defining which parts of a data base have inquiry capability and which users have access to each part. An additional level of protection is available through the specification of valid inquiry users of each logical database.

A brief description of the DMS/INQUIRY statements follows:

ATTACH

Allows the user to combine an embedded structure with its owner to establish automatic looping between the two structures.

CLEAR

Discards DEFINE items, VIRTUAL items, and/or GENERATEd subsets.

DEFINE

Allows INQUIRY text to be assigned a name. When INQUIRY sees the define name, it replaces the define name with the associated text.

DETACH

Separates an embedded structure from its owner to prevent automatic looping between the two structures.

DISPLAY

Allows items of a selected record to be displayed.

EDIT

Allows a previous INQUIRY statement to be modified without requiring the entire statement to be re-entered.

GENERATE

Creates a temporary subset of a data set. The temporary subset can be referenced by other INQUIRY statements.

HELP

Displays the syntax and semantics for each INQUIRY statement.

NEXT

Causes INQUIRY to resume record selection and item display.

OPTIONS

Allows INQUIRY options to be displayed or altered.

PRINTER

Allows the attributes of the line printer file to be displayed or altered.

QUIT

Terminates the INQUIRY session.

RECALL

Retrieves the text of a prior INQUIRY statement.

REPEAT

Causes re-execution of a previous INQUIRY statement.

RESTORE

Allows previously SAVEd text to be retrieved.

SAVE

Stores the text of DEFINE items, VIRTUAL items and GENERATEd subsets in a file on disk. The SAVEd text can be reloaded and used during subsequent INQUIRY sessions.

SELECT

Locates records which satisfy the selection criteria provided by the user.

SET

Modifies or deletes the text of the most recently entered DISPLAY, REPEAT or SELECT statement for a given data set.

SHOW

Displays all or selected portions of the database description and may also be used to display the most recently entered INQUIRY statement.

SORT

Allows a user to control the amount of core and disk used by the SORT option.

TERMINAL

Allows the attributes of the terminal file to be displayed or altered.

VIRTUAL

Allows new items to be defined which are functions of other items.

OPERATING INSTRUCTIONS

ACCESSIBILITY

The DMS/INQUIRY facility provides the ability to specify which portion of a data base is to be visible to any particular user. The user may be allowed access to the entire data base or any single logical data base. The method for specifying accessiblity is defined in Appendix A.

TERMINAL USE

Log-on procedures are defined in Appendix A. Once log-on has been accomplished the user may then enter inquiry statements.

The terminal attributes are known to DMS/INQUIRY allowing appropriate output formatting without the user having to specify them at run time.

Terminal input is received by DMS/INQUIRY one line at a time with the line width being determined by the type of terminal and the TERMINAL WIDTH option. The maximum line width may not exceed that of the terminal being used.

In some cases an inquiry statement may require entering an input which exceeds the line width capacity of the terminal. For this reason, the system maintains a current text buffer capable of holding several lines of input. While the text of some statements is processed directly in the input buffer, the text of other statements is moved to and then processed from the current text buffer. Those statements whose text is moved to the current text buffer are:

DISPLAY
SELECT
GENERATE
SET
DEFINE
VIRTUAL

For the above statements, if the text associated with the statement exceeds the maximum that can be entered in one line, a % can be entered just ahead of the end-of-message character. When the system responds with a #%, additional input can be entered. This can be repeated until the complete statement has been entered.

If it is desired to enter an input and have the system "remember" the text but not process it, the line of input (or last line of input) may be ended by a %% just ahead of the end-of-message. If the input is incorrect, it can be corrected with an EDIT statement (See EDIT). The REPEAT statement can be used to cause the system to process or reprocess the contents of the current text buffer, while the SHOW statement can be used to display the contents of the current text buffer.

The current text buffer can also be loaded with previously entered text (See RECALL).

Certain abbreviations are allowed for the DMS/INQUIRY statements. Below is a list of statements; the underlined portions represent the minimum recognized abbreviation.

```
ATTACH               NEXT                   SAVE
--                   -                      --

CLEAR                OPTIONS                SELECT
-                    -                      -

DEFINE               PRINTER                SET
---                  -                      ---

DETACH               QUIT                   SHOW
--                   -                      --

DISPLAY              RECALL                 SORT
-                    ---                    --

EDIT                 REPEAT                 TERMINAL
-                    -                      -

GENERATE             RESTORE                VIRTUAL
-                    ---                    -

HELP
-
```

Some input commands may produce an excessive amount of output. This output, shown one page at a time, can be discontinued by entering a non-null response after a page has been displayed.

The system will then respond with a "#" indicating that it is ready for user input.

BASIC ENTITIES

Letters consist of upper case A thru Z and lower case a thru z. All lower case letters are translated to upper case except when used in alpha-literals.

Digits consist of 0 thru 9.

Names, unsigned integers, numbers and alpha-literals are formed as specified in the DASDL Product Specification, 2219 0433.

Names include those of accessible data base items, DMS/INQUIRY keywords, all possible valid abbreviations of the verbs, virtual items, define items, temporary set and file names.

Alpha-literals need not, in some circumstances, be contained within quotes. When the OPTION QUOTES = FALSE, alpha-numeric strings will be considered as alpha-literals if they are not the same as any data base item, virtual, define or temporary set names.

DATA RELATIONSHIPS

The data relationships of a data base can be represented by a hierarchy or tree structure. Each data set with its embedded data sets is said to be a tree structure. For a given data set Y, each possible ordered set X of elements X_i where

$X_1 = Y$

X_i is embedded in X_{i-1} for $1 < i <= n$

X_n has no embedded data set

is said to be a branch of the tree structure for Y.

UNIQUENESS OF NAMES

Names of database items not included in the user's logical data base may be used to identify other DMS/INQUIRY objects.

Define formal parameter names may be the same as any other name except any define formal parameter in the same parameter list.

File names may be the same as any other name.

DMS/INQUIRY keyword names may be the same as a data base item name. When there is a conflict in interpretation of use DMS/INQUIRY assumes the name to be of a data base item. A "#" preceding the name may be used to cause DMS/INQUIRY to assume the reserved word use.

With the exceptions above, temporary set, virtual and define names must be unique.

BASIC INQUIRY

The process of producing information on a user's terminal consists of two functions:

a. Selection of those records of interest.

b. Displaying user specified information from these records on the user's terminal.

For example, suppose a company had a data base which contained information on the company's employees. If the user enters

```
SELECT ROOM = 421
```

the system would locate the first record for an employee residing in Room 421. If the system fails to locate at least one such record, it will type out

```
# NOT FOUND
```

If the system does locate a record, it will type out only the #. If the user now wanted some information, such as the person's name and extension number, he would enter

```
DISPLAY NAME, EXTENSION
```

The system would respond as follows:

```
NAME = JONES  JOHN D

EXTENSION = 2364
```

If the user wishes to know if more than one person resided in Room 421, he could then enter

```
NEXT
```

If another record existed, the system will respond by typing a #. If no more records exist, the system will respond with a # NO MORE.

SELECT

The text, ROOM = 421, in the example

```
SELECT ROOM = 421
```

is a selection-condition.

The simplest form of selection-condition is in the form:

```
name  relational-operator value
```

Example:

```
     SALARY > 500

     JOBCLASS = "PROGRAMMER"

     HOURS-WORKED < 40
```

In the above examples, the characters >, =, < are relational operators. The following table indicates allowable relational operators.

Operator	Alternate Spelling	Meaning
=	EQL	EQUAL TO
>	GTR	GREATER THAN
<	LSS	LESS THAN
<=, ¬>	LEQ	LESS THAN OR EQUAL TO
>=, ¬<	GEQ	GREATER THAN OR EQUAL TO
¬=	NEQ	NOT EQUAL TO

The logical operator NOT together with parenthesis can be used to specify an "all except" condition.

Example:

```
     SELECT NOT(JOBCLASS = "MANAGER")
```

would select all people except those who are managers.

The logical operators AND and OR can be used to combine simple conditions into more complex conditions.

Example:

```
     SELECT DEPT=6700 AND JOBCLASS="MANAGER"
     SELECT DEPT=6700 OR DEPT=6800
```

Care must be taken when using the AND and OR both within the same condition.

Example:

```
     SELECT DEPT=6700 AND JOBCLASS="PROGRAMMER" OR SALARY > 500
```

This select statement woulo locate all records in DEPT=6700 where JOBCLASS was programmer. Also it would locate all persons whose

SALARY>500 regardless of department. The reason for this is that AND is always performed first.

Parenthesis can be used to control which conditions are ANDed with other conditions. For example:

```
SELECT DEPT=6700 AND (JOBCLASS="PROGRAMMER" OR SALARY>500)
```

This select statement would locate only those people in DEPT=6700, who are programmers or whose salary exceeds 500.

QUALIFICATION

A data base consists of one or more data sets. Each data set consists of a number of records where each record consists of a number of items. In the preceding examples, it was not necessary to name the data set containing the records to be selected. The DMS/INQUIRY will attempt to determine the data set name by analysis of the names appearing in the item list and/or the selection condition. This is not possible in all cases. For example, consider the following:

```
INVENTORY DATA SET
  UNIT-PRICE
  PART-NO

ORDERS DATA SET
  UNIT-PRICE
  PART-NO
```

In this case two data sets have records with the same item names. A request to

```
SELECT PART-NO = 1234
```

would cause DMS/INQUIRY to respond with:

```
# WHICH DATA SET?

   1. INVENTORY
   2. ORDERS
```

The user would then respond by entering either 1 or 2.

The user can designate the correct data set at the time the DISPLAY is entered.

Example:

```
SELECT PART-NO OF INVENTORY=1248
DISPLAY UNIT-PRICE OF INVENTORY
```

QUALIFYING SELECTION EXPRESSIONS

To determine the object dataset of a selection expression, the left sides of all relationals in the expression are analyzed to produce an object dataset. The VIA structure, if specified, is included in this analysis. In the case where this analysis does not yield one unique possibility, DMS/INQUIRY will attempt to choose from the multiple pcssibilities by applying:

1. its knowledge of the structure of the database, and

2. its knowledge of past references to the database through DMINQ statements.

CHECK CURRENT HIERARCHY

The object dataset of the last selection expression processed will be used in the first attempt at qualification. DMS/INQUIRY will first assume that the present selection refers to:

1. the same dataset as the last selection expression, or

2. a dataset embedded in that dataset.

Should both of these criteria fail to reduce the number of possible object datasets, DMS/INQUIRY will proceed by considering the parent dataset of the last dataset selected. The same test as above will be applied. The dataset, then its immediate descendant datasets, will be checked to determine whether they can qualify the selection expression. This process of checking the current "branch of the tree" continues until:

1. one of the criteria causes the number of possible object datasets to decrease, or

2. the disjoint level is reached, and fails to reduce the number of possible object datasets.

If DMS/INQUIRY has reduced the number of possible object datasets at any of the steps above, it will ask the user to choose from among those possibilities which remain. If a single dataset remains, DMS/INQUIRY will use it as the object dataset of the selection expression in question.

CHECKING PAST DATASET REFERENCES

Should the above procedure fail to produce one unique object dataset, DMS/INQUIRY will choose as the object dataset the dataset which was most recently referenced in a DMS/INQUIRY statement. A reference to a dataset consists of using any expression, which, in order to be evaluated, requires that dataset to have a valid current record.

ASKING THE USER

If the above two methods fail to yield one unique object dataset, the user will be asked to choose from among the possibilities.

QUALIFYING ITEMS IN SELECTION EXPRESSIONS

Once DMS/INQUIRY has determined the object dataset of a selection expression, any item names appearing in multiple datasets must be qualified. The procedure of qualification is similar to that for qualifying selection expressions, as it uses both the current hierarchy and the record of past references to the database. DMS/INQUIRY will first try the assumption that:

1. the item is in the object dataset of this selection expression,

then try the assumption that:

2. the item is in a dataset in the current "branch of the tree".

If these criteria fail to yield a unique parent dataset for the item, DMS/INQUIRY will choose as the possible parent dataset the possible dataset which was most recently referenced by a DMS/INQUIRY statement. If the above also fails, the user will be asked to supply explicit qualification.

QUALIFYING FUNCTIONS

The process of determining the object dataset of a function selection condition is again similar to the two qualification resolution processes explained above. The basic difference occurs when the function does not reference an embedded dataset at the next lower level from the current object dataset. When DMS/INQUIRY is unable to choose an object dataset on this basis, it will immediately assume that the object dataset of the function lies outside the current "branch of the tree" and will proceed to the checking of past references to datasets.

STRUCTURE DESIGNATION

For each data set in a data base, there may be a number of sets and subsets associated with it. The system will always attempt, if possible, to locate selected records by utilizing index sets. It will never automatically utilize subsets to locate records as subsets may not locate all the desired records.

The user may, however, force DMS/INQUIRY to locate records through a given structure. For example, given a data set with sets and subsets as follows:

```
     PERSONNEL DATA SET
       NAME ALPHA (17)
       MANNUM NUMBER (5)
       DEPT NUMBER (4)

     EMPNUMSET SET OF PERSONNEL KEY MANNUM
     DEPT6700 SUBSET OF PERSONNEL WHERE DEPT=6700 KEY NAME
```

The statement

```
     SELECT MANNUM=28901
```

would use the AUTOMATIC set EMPNUMSET to locate the record, while

```
     SELECT NAME="DOE, JOHN"
```

would cause the file PERSONNEL to be searched. The system would not use DEPT6700, as this structure is a subset and may not locate all of the requested records.

The user may force the system to locate records in the order of a particular structure. For example:

```
     SELECT PERSONNEL AT MANNUM>500
```

would cause DMS/INQUIRY to locate records through the data set in physical storage order.

```
     SELECT EMPNUMSET
```

would cause DMS/INQUIRY to locate all records sequentially through the automatic set EMPNUMSET.

```
     SELECT DEPT6700 AT NAME="DOE, JOHN"
```

would cause the system to select records through the subset DEPT6700 (which for this example would locate only those JOHN DOE who are in department 6700).

DISPLAY

Once a record has been selected by the user, he may wish to display one or more items within the record. This is accomplished by utilizing the DISPLAY statement.

Example:

```
     SELECT ROOM=421
     #
```

```
DISPLAY NAME
  *NAME=SMITH  JOHN
DISPLAY EXTENSION
  *EXTENSION=2364
DISPLAY DEPT
  *DEPT=6145
```

Note: * denotes system response but in actual use does not appear on the user's terminal.

As indicated by the above example, once a record has been selected, one or more DISPLAY statements are allowed.

More than one item can be displayed by entering a list of desired items.

Example:

```
SELECT ROOM=421
#
DISPLAY NAME, EXTENSION, DEPT
 *NAME=SMITH  JOHN
 *EXTENSION=2364
 *DEPT=6145
```

When displaying an item list, a comma must appear between the names of the items in the item list.

If it is desired to display all items in a record, the user can enter ALL.

Example:

```
DISPLAY ALL
```

This will display all items in the most recently-selected structure.

COMBINED SELECT/DISPLAY

The process of record selection and item display can be combined into a single inquiry statement by adding a selection condition to the DISPLAY statement.

```
DISPLAY NAME IN ROOM=421.
```

In this example the word IN separates the item list of the display from the selection condition. In addition to the word IN, the words AT and WHERE can be used as separators, as well as the special character @.

Examples:

```
     DISPLAY SALARY AT MAN-NUMBER=2890
     DISPLAY ROOM WHERE EXTENSION=2364
     DISPLAY EXTENSION @ ROOM=421
```

If structure designation is required, this designation can be inserted just ahead of the separator.

Example:

```
     DISPLAY UNIT-PRICE USING ORDERS WHERE PART-NO=1248
```

PROCESS TERMINATION

The commands SELECT/DISPLAY, GENERATE, REPEAT, SHOW and HELP may generate an excessive amount of output or require an excessive amount of system time. The processing and/or output of these statements may be terminated by entering a non-null response after a page full of information has been displayed.

The status of DMS/INQUIRY after such a termination is not defined.

OUTPUT CONTROL

When using SELECT as a statement, the user gets one record at a time. To obtain additional records which satisfy the criteria of the selection-condition, the NEXT statement must be used.

The system's response to a combined select/display inquiry statement, however, is to display the information for all records selected without further intervention by the terminal user, unless the terminal is a CRT device. For CRT devices the system will display that amount of information which will fill the screen then wait. At this point the user can have the system present another screen full of information by entering "NEXT" or an end-of-message. If the user enters anything else the inquiry is cancelled.

The output produced by an inquiry may become excessive for printing on the terminal. The amount of uninterrupted output can be controlled on a statement basis.

Example:

```
     DISPLAY 3 NAME IN DEPT=6700
```

The system will display output for 3 records, and then wait. Entering NEXT will either display the remaining output, or display the output for 3 more records and then wait again.

An alternate method for entering NEXT is to enter only the end-of-message.

EMBEDDED STRUCTURES

In a data base it is possible to have datasets appear as items of records of another data set. For example:

```
D  DATA SET
  D1  NUMBER
  D2  ALPHA
  E    DATA SET
    E1 NUMBER
    E2 ALPHA
  D3  NUMBER
```

In this example, E is an embedded data set of D and D is the parent of E. D is also considered a disjoint dataset as it has no parent. Another case of embedded structures is as follows:

```
D  DATA SET
  D1  NUMBER
  D2  ALPHA
  SE  SUBSET OF E KEY E1
  D3  NUMBER
E  DATA SET
  E1  NUMBER
  E2  ALPHA
```

The subset SE is an embedded subset of D which points at some or all of the records of the disjoint dataset E.

Several cases exist for inquiry against embedded structures. For each of these cases, the following dataset is assumed:

```
D  DATA SET
  D1
  D2
  D3
  E   DATA SET
    E1
    E2
    E3
    F   DATA SET
      F1
      F2
```

D is a data set containing items D1, D2, D3 and the embedded data set E. E is an embedded data set containing items E1, E2, E3 and the embedded data set F. F is an embedded data set containing items F1 and F2.

DISPLAY ITEMS OF EMBEDDED STRUCTURES

Example:

```
     SELECT D1=3, THEN DISPLAY E1,E2 WHERE E3>5
```

This example will select all D records where D1=3. For each D record selected, the system will display E1 and E2 for each E record where E3>5.

If structure designation were required, the example would be

```
     SELECT D WHERE D1=3, THEN DISPLAY E1,E2 USING E WHERE E3>5
```

Items of F could be displayed as follows:

```
     SELECT D1=3, THEN SELECT E3>5, THEN DISPLAY F1 AT F2=7.
```

One could display items from each data set.

Example:

```
     DISPLAY D2 WHERE D1=3, THEN DISPLAY E1 WHERE E3>5, THEN
       DISPLAY F1 AT F2=7
```

The output appearing on the terminal would be

```
     D2 = 28
     E1 = 4
     F1 = 7
     E1 = 6
     F1 = 3
     F1 = 10
     F1 = 14
     E1 = 7
     E1 = 8
     F1 = 18
     D2 = 30
     E1 = 8
     F1 = 4
     F1 = 8
```

(until there are no more D records).

For this example the item is displayed before the system attempts to locate a record at an embedded level.

Since each record must be located before an attempt is made to locate a record of an embedded structure, the item list in a DISPLAY may reference items of records at previous levels.

Example:

```
     SELECT D1=3, SELECT E3>5, DISPLAY D2, E1, F1 WHERE F2>0
```

Note: Either a comma, or THEN, or both must be used to separate clauses in a complex inquiry statement.

This example differs from the preceding example in that a DISPLAY will occur only if a record exists at all levels.

The above example may require qualification of the displayed items by the data set name, as the item name may exist in the records of more than one dataset; i.e.,

```
. . . DISPLAY D2 OF D, E1 OF E, F1 OF F
```

OUTPUT CONTROL

The system, in response to an inquiry, will output results until all records selected are displayed (or the screen of a CRT device is full).

The amount of uninterrupted output can be controlled on a statement basis. For example:

```
SELECT D1=3 THEN DISPLAY 3 E1 WHERE E2=7
```

For this example the 3 after the DISPLAY will cause the system to output items for 3 records of E, then wait. Output can be continued by entering NEXT.

The use of the output control numbers can also be used to selectively bypass some of the output.

Example:

```
SELECT D1=3 THEN DISPLAY 3 E1 AT E2=7
```

The system will display E1 for 3 records of E and stop. If the user then enters NEXT D the system will

a. Cancel output for E records of the current D record.

b. SELECT the next D record.

c. Output the first 3 E records selected.

Output control can be imposed at each level.

Example:

```
DISPLAY 3 D1 AT D2=5 THEN DISPLAY 2 E1 AT E2=7
```

The system will stop after displaying 2 E1's. It will also stop after oisplaying 3 D1's. This may require entering NEXT twice to resume output.

ARITHMETIC EXPRESSIONS

In the previous examples the selection conditions presented were all in the form

```
     name relational-operator value
```

e.g.

```
     D1 > 5
     D3 = 7
```

The system, in addition to numeric and alphabetic literals, also allows arithmetic expressions in selection conditions, e.g.

```
     D1 > D2+D3
     D4 < ((D2+D3)*D4)/D5
     D7 > D8
```

The arithmetic operators allowed in these arithmetic expressions are:

+	add
-	subtract
*	multiply
/	divide
DIV	integer divide
MOD	modulus

The rule as to what identifiers may appear in an arithmetic expression is: all the terms appearing in the expression must be capable of yielding a value when the record containing the item to the left of the relational operator is loaded.

For example, given the database:

```
     D
      D1 NUMBER
      D2 NUMBER
      D3 NUMBER
      E  DATA SET
       E1 NUMBER
       E2 NUMBER
```

LEGAL examples

```
     SELECT D1 > D2
     SELECT D3 > (D1+D2)
     SELECT VIA D, SELECT E2 > E1+D1
```

ILLEGAL examples

```
     SELECT D1 > D1+E2
```

Note: A value for E2 cannot be derived as no E record has been selected.

FUNCTIONS

Sometimes it is necessary to select a master record as a result of some function of its embedded dataset.

BOOLEAN FUNCTIONS

For example, suppose for each employee there is an embedded dataset containing job history information. Suppose it is desired to know the names of everyone in DEPT=6700 who has been an ENGINEER.

Example:

```
     DISPLAY NAME WHERE DEPT=6700 AND ANY(JOBCLASS="ENGINEER")
```

The condition

```
     ANY(JOBCLASS="ENGINEER")
```

is a Boolean function as it yields a truth value

```
     TRUE or FALSE.
```

The system will select only DEPT=6700 records. For each of these selected records, the system will search the employee's job history looking for a job class of engineer. If any are found, the system will display the employee's name.

The condition within the parenthesis may be a complex condition.

Example:

```
     ANY(JOBCLASS="ENGINEER" AND TIMEHELD=18)
     ANY(JOBCLASS="ENGINEER" OR JOBCLASS="MANAGER")
```

The second form of the Boolean function is ALL. For example,

```
     DISPLAY NAME WHERE DEPT=6700 AND ALL(JOBTITLE="ENGINEER")
```

This example would display the names of all personnel in DEPT=6700 who have only held positions of ENGINEER.

ARITHMETIC FUNCTIONS

Suppose there is a BANK data set. Each bank record has an embedded STOCK data set which contains information about stock the bank owns. A list of banks which own more than $1,000,000 in stock is required.

Example:

```
     DISPLAY BANK-NAME WHERE SUM(STOCK-VALUE) > 1000000
```

The expression

```
     SUM(STOCK-VALUE)
```

is an arithmetic function. In addition to SUM, the system allows AVERAGE, MAX, MIN, SUM OF SQUARES, MEAN SQUARE, VARIANCE and STANDARD DEVIATION (see FORMAL DEFINITION (Section 3) for details).

In the above example, the STOCK-VALUE items for all STOCK were summed, then compared with the specified value. A condition can be specified which allows selecting only some of the records to be summed.

Example:

```
     DISPLAY BANK-NAME
       WHERE SUM(STOCK-VALUE WHERE TENDOR="ABC") > 1000000
```

This example would display bank-names for those banks which owned more than $1,000,000 in Company ABC.

A special type of arithmetic function is the COUNT. COUNT will count the number of occurrences of some condition within the embedded dataset. For example:

```
     DISPLAY BANK-NAME WHERE COUNT(TENDOR="ABC") > 9
```

This example would display the names of banks owning more than 9 stock certificates in ABC Company.

FUNCTION DISPLAY

Arithmetic functions (but not Boolean functions) may also appear in the display item list.

Example:

```
DISPLAY BANKNAME, SUM(STOCK-VALUE) FOR STATE = "NEW-YORK"
```

When using FUNCTIONS within a selection condition or display list, it must be remembered that at least one term associated with the function must be in a dataset embedded within the object dataset of the SELECT/DISPLAY expression. Given a data base which contains:

```
D  DATA SET
  D1
  D2
  E DATA SET
    E1
    E2
```

Legal:

```
DISPLAY D1, SUM(E1) AT D2>0 AND AVG(E2)>50
```

Not legal:

```
DISPLAY E1, SUM(D2)
```

as D2 is not in a dataset embedded in E.

One can always

```
DISPLAY AVG(SALARY), MIN(SALARY)
```

where salary is an item of the disjoint dataset PERSONNEL.

Caution: To compute the values for the various functions it may be necessary for the system to access every record of the dataset involved; thus it may take some large amount of time to produce a result.

VIRTUAL ITEMS

It is sometimes desirable to compute and/or display a value which is not physically held in the database. For example it may be desirable to display a person's weekly salary but only HOURS-WORKED and PAY-RATE is held in the dataset.

The concept of Virtual items is implemented to handle this. For example the system allows

```
     VIRTUAL SALARY = HOURS-WORKED*PAY-RATE
```

The system would remember SALARY and treat it as if SALARY were an actual item. The user can then use this virtual item in a display item list or selection condition.

Example:

```
     DISPLAY NAME, SALARY IN DEPT=6700
     DISPLAY NAME, DEPT AT SALARY>400.00
```

Any form of arithmetic expression is allowed to the right of the "EQUAL" sign in the virtual text. The rule is that the items must be capable of yielding a result when the virtual item is referenced. For example, given the database which contains

```
     D DATA SET
       D1
       D2
       E DATA SET
         E1
         E2
```

Given:

```
     VIRTUAL DX  = D1+D2
     VIRTUAL DE  = E1+D1
```

Note: Since D exist when E exist, the virtual DE can contain items of D and E.

Legal:

```
     DISPLAY DX AT D2 = 50
```

Not legal:

```
     DISPLAY DE AT D2 = 50
```

Note: Since E does not exist when D exist the E item (E1) cannot be part of the virtual DE. A record not selected error will occur.

The arithmetic expressions associated with a virtual may also contain arithmetic functions. For example

```
     VIRTUAL DX = D1+SUM(E1 AT E2>20)
     VIRTUAL DY = AVG(E2)
```

It is advantageous in terms of system efficiency to associate arithmetic functions with virtual identifiers. If a function appears in a display item list or selection condition, the system must evaluate the function each time the function is referenced. However, if the function is associated with a virtual, the system "remembers" whether a function has been evaluated and, if so, the value of the function.

Note: The system also recognizes when the value of such a function becomes meaningless and will re-evaluate it when necessary.

UNDEFINED ITEMS (values print as hyphens)

There are several cases where references can be made which can result in undefined situations. These cases may occur either in a display list, a selection condition, or an arithmetic expression. The undefined cases are as follows:

1. The value of an item is NULL (as defined in DASDL).

2. The value of a function is undefined. For example:

   AVG(E2 where E1>100)

   and there are no records where E1>100.

3. Variable format records are used and a record was loaded in which the specified item does not exist.

4. The process of evaluating an arithmetic expression would have resulted in division by 0, or an attempt was made to generate a number exceeding the hardware capacity (integer overflow, exponent overflow, or exponent underflow).

When an undefined situation is encountered in displaying an item, two hyphens are printed in lieu of a value.

The occurrence of an undefined situation becomes slightly more complex when encountered within a selection condition. For example

```
SELECT NOT(A=B) OR C>50
```

if either A or B is undefined the selectional expression

```
NOT(A=B)
```

is false regardless of the presence of the NOT and regardless of the relational operator used. The record would be selected only dependent on the truth value of the relational expression C>50. However if C was also undefined, for the current record being looked at, then the record would not be selected.

UNQUOTED ALPHA-LITERALS

When entering selection conditions in various inquiry statements, relationships in the form:

```
    <ALPHA-ID> <RELATIONAL> <ALPHA-LITERAL>
```

are encountered. ALPHA-LITERAL is defined by a quote ("), followed by one or more characters, followed by an ending quote.

Example: "THIS IS AN ALPHA-LITERAL"

One of the most frequent terminal user errors is to fail to bracket ALPHA-LITERALS by the beginning and ending quotes.

The INQUIRY system will recognize alphanumeric strings as alphanumeric literals even though quotes are not used.

Note: An alphanumeric string is defined as a series of characters containing only lowercase A-Z, uppercase A-Z and the numeric characters 0-9. Special characters are not allowed.

However, since the use of unquoted alpha-literals can be misinterpreted by the system (discussion follows), the following option can be utilized.

```
>--- OPTION QUOTES ------------------------------------------------>#
                        |                                         |
                        |------------------- TRUE ------->|
                        |             |    |-- FALSE ------>|
                        |---- = ---->|
```

1. If quotes or quotes = TRUE, then all alpha-literals must be quoted. This is the default set on initializing INQUIRY.

2. If quotes = FALSE then

   a. An alphanumeric string will be recognized as an alpha-literal when used in the proper context.

   b. Alpha-literals containing special characters ($%&) and blanks must be quoted.

   c. Even if quotes = FALSE, quoted alpha-literals are allowed.

As previously stated, the system may misinterpret the user intent when unquoted alpha-literals are used. For the following discussion and examples, A represents an alpha identifier in the data set D.

Example 1:

```
SELECT A = 1234
SELECT A = 12A856
```

An unquoted alpha-literal which starts with numeric characters will not be misinterpreted.

Example 2:

```
SELECT A = X13
```

X13 is the name of the data base item. In this case the system cannot know if the user wanted the value of the identifier X13 or the alpha-literal "X13". The system will assume that request was for the identifier.

Example 3:

```
DEFINE X13 = RST
```

and then

```
SELECT A = X13
```

The system will see this as if the user had entered

```
SELECT A = RST
```

If the alphanumeric string is the name of a define, the define will be expanded and the text of the define used.

Two safeguards exist which can be used to determine if the system will recognize the user intent.

a. If a SHOW X13 is entered and this results in an unknown identifier or invalid option error, then X13 will be accepted as an alpha-literal.

b. Given that a selection expression has yielded unexpected results, entering a

```
RECALL D
```

where D is the data set name, will indicate what was scanned by the system as any unquoted alpha-literal will be redisplayed as a quoted alpha-literal.

OUTPUT FORMATTING

The display of data on the terminal is under the control of the setting of the TERMINAL FORMAT attribute HEADING, TAB and SINGLE. (See TERMINAL verb in section 3).

The default setting when more than one record is being displayed is HEADING.

The default setting when only one record is being displayed is TAB.

HEADING

This format attribute can be set by entering

```
        TERMINAL FORMAT HEADING
```

The output display will be in the form:

```
N1     N2     N3
V11    V12    V13
V21    V22    V23
```

where N1, N2 and N3 are the names of the items and

V11 is the value of N1 in record 1
V12 is the value of N1 in record 2
etc.

In some cases the size of the display list will be such that all the names and/or values will not fit on one line. For this case the output will be in the form:

```
 1:N1     N2     N3
 2:N4     N5     N6
 1:V11    V12    V13
 2:V14    V15    V16
 1:V21    V22    V23
 2:V24    V25    V26
```

where N1 thru N6 are the names of the items and the Vij are the values of the items.

Association between names and values is made by line number and position within the line.

The number of title lines and value lines for a single record may be such that, for a crt device, the screen would not be large enough. The first time the system detects this it will automatically reset the FORMAT attribute to TAB.

TAB

This format attribute can be set by entering

TERMINAL FORMAT TAB

The output display will be in the form:

```
 N1=V1    N2=V2    N3=V3
 N4=V4    N5=V5
```

The intention of the TAB format is for the case where the display list output cannot fit on one screen. Each name = value combination will start at a TAB position where the TAB stops are every 5th position starting at character position 2.

SINGLE

This format attribute can be set by entering

TERMINAL FORMAT SINGLE

The output display is in the form:

```
     N1=V1
     N2=V2
     N3=V3
```

That is one name and value per line. The intention of the SINGLE attribute is for those terminals which have small line widths.

DMS/INQUIRY DEFINITION

STATEMENTS

ATTACH

```
ATTACH -------------- <data-set-id> -------------------------->#
--
```

1. ATTACH is used to combine a structure with its owner to provide automatic selection looping between the two structures.

2. Data-set-id, e.g., Xi, must be an embedded structure already specified in a SELECT/DISPLAY command (this implies that Xi-1 has also been previously specified).

3. Subsequent to this command, whenever a record is selected for Xi-1, a record will be automatically selected for Xi. In addition, whenever Xi is exhausted, Xi-1 will be automatically selected. Thus, this command "joins" the two structures.

4. Note that Xi is by default attached to Xi-1, if both structures are given in the same SELECT/DISPLAY command and are not subsequently DETACHed.

5. If Xi is already attached to Xi-1, this command is ignored.

CLEAR

```
CLEAR ---------- DECLARATIONS ---------------------------------->#
-       |                                       |
        |------- GENERATES -------------------->|
        |                                       |
        |------- VIRTUALS --------------------->|
        |                                       |
        |------- DEFINES ---------------------->|
        |                  |                 |  |
        |                  |-- <define-id> ->|  |
        |                                       |
        |------- <id> ------------------------->|
```

1. The CLEAR statement causes the system to clear the text declared by a CEFINE, VIRTUAL, or GENERATE statement.

2. The keyword DECLARATIONS causes the identifier and text, established by all VIRTUAL, DEFINE and GENERATE statements, to be removed from the inquiry system's set of identifiers.

3. If the keyword VIRTUALS, GENERATES, and/or DEFINES is used, the system will clear all text of all identifiers declared by that verb. For the special case

   CLEAR DEFINES <define-id>

   only the <define-id> will be cleared.

4. If <id> is used, it must be the name of an identifier established by a DEFINE, VIRTUAL, or GENERATE statement. This <id> and its text will be removed from the inquiry system's set of identifiers.

DEFINE

Option 1:

```
DEFINE ------- <define-id> --- = --- <text> ------------------>#
---
```

Option 2:

```
                                        |<- , --|
                                        |       |
DEFINE ------ <define-id> --- ( ---<par>--- ) -- = -- <text> -->#
---
```

1. DEFINE establishes the given <define-id> as an abbreviation for the given text.

2. Option 2 is the parametric define. It allows a portion of text to be defined, with the remaining portion being supplied at the time the <define id> is referenced. The <par> is a name meeting uniqueness requirements. For example:

   ```
   DEFINE GET(A,B) = DISPLAY NAME AT A = B
   ```

   The identifiers A and B are formal parameters. When the <define-id> is referenced, the text which appears in the position of the formal parameters is inserted into the define text at the position where the formal parameters appear. For example:

   ```
   GET(EXT,2364)
   ```

   will cause the system to "see" the statement:

   ```
   DISPLAY NAME AT EXT = 2364
   ```

3. References to defines may appear anywhere a name may appear. However, file names and formal parameters are not treated as references to define-ids.

4. The text of the define may contain references to other defines. A define may not contain a reference to itself and the referenced define text may not contain any define references.

5. Text for a previously established define may be changed only by:

   a. CLEARing <define-id> and re-entering the DEFINE, or:

b. RECALLing DEFINES <define-id>, EDITing and REPEATing.

DETACH

```
DETACH --------------- <data-set-id> --------------------------->#
--
```

1. DETACH is used to separate a structure from its owner in order to prevent automatic selection looping between the two structures.

2. The <data-set-id>, e.g., Xi, must be an embedded structure already specified in a SELECT/DISPLAY command (this implies that Xi-1 has also been previously specified).

3. Subsequent to this command, whenever Xi-1 is selected, Xi will be marked as having no current record selected; whenever Xi is exhausted, the system stops with an appropriate message.

4. Note that Xi is by default attached to Xi-1 if both structures are given in the same SELECT/DISPLAY command. It is necessary to DETACH Xi to prevent automatic selection looping between the two structures.

5. If Xi and Xi-1 have been previously specified, but in different SELECT/DISPLAY commands, and if they have not been subsequently ATTACHed, they are by default DETACHed.

6. If Xi is already detached from Xi-1, this command is ignored.

DISPLAY (Current)

```
DISPLAY ------------ <display-list> ----------------------------->#
-           |                                                   |
            |                                                   |
            |                                                   |
            |--- ALL ------- OF --- <data-set-id> ---->|
                         |                                      |
                         |                                      |
                         |------------------------------>|
```

1. Display on the user's terminal the values of:

   a. Items specified by the <display list>; or

   b. ALL items of <data-set-id>, if given; or

   c. ALL items of the most recently-selected structure.

2. All items to be displayed must be contained in structures for which there are current records selected. Referencing other items will result in an error. Items need not all appear in the same branch of the data base.

3. Note that this command is a special case of the general SELECT/DISPLAY command. No selection conditions are given, and no change is made in the status of any structure, nor in the selection conditions or display-lists, if any, attached to any structure.

4. The ALL option means to display all items of the record. If the OF <data-set-id> is not specified, then the items of the record last selected will be displayed.

EDIT

```
EDIT------- <delim> --- <text1> --- <delim> -------------------->#
-       |                                     |             |
        |                                     |- <text2> -->|
REPEAT >|
-
```

1. EDIT allows modification of the current text buffer of the DMS/INQUIRY system (see SHOW and RECALL).

2. EDIT searches for the literal appearance of <text1> in the current text buffer.

3. If <text2> is not specified, then <text1> is eliminated from the current text.

4. If <text2> is specified, then <text1> is replaced by <text2>.

5. The delimiter <delim> can be any special character not appearing in <text1>.

6. The REPEAT verb can be followed by the EDIT syntax; this is identical to EDIT... followed by REPEAT.

GENERATE

```
GENERATE --- <temp-set-id> ---- = --- <select-spec> ----------->#
-                                  |                      |
                                   |- <temp-set-exp> --->|
```

```
<temp-set-exp>
```

```
<temp-set-id> --------------------------------------------------->#
               |                                 |
               |--- AND ------ <temp-set-id> -->|
               |           |
               |--- OR --->|
               |           |
               |---  +  -->|
               |           |
               |---  -  -->|
```

1. Generates a temporary subset of structure X:

   a. satisfying the given <select-spec>, or
   b. equal to another temporary subset, or
   c. equal to a set function of two other temporary subsets.

2. The temporary subset so generated may be subsequently referenced by the given <temp-set-id>. It may be used wherever X may be used.

3. All temporary subsets used must be subsets of structure X. The two <temp-set-id>s in the <temp-set-exp> must be different.

4. The permissable set functions are:

   AND : set intersection
   OR : set union
   - : set difference
   + : exclusive OR

5. For details of <select-spec> see SELECTION CRITERIA.

6. X must be a disjoint structure.

HELP

```
HELP ---------------------------------------------------------->|
             |                                         |
             |                                         |
             |---- VERB ------------------------------>|
             |                     |  |                |
             |--- <verb-name> ---->|  |-- SEMANTICS -->|
             |                     |
             |--- <syntax-item> -->|
```

1. The HELP verb supplies information to a user on how to use DMS/INQUIRY.

2. If HELP is entered following an error output, the system will display an explanation of the error encountered.

3. HELP VERB will display a list of all inquiry verbs.

4. HELP <verb-name> will display the syntax diagram of the verb.

5. HELP <syntax-item> will display the syntax of the syntax item. Note: a "syntax-item" is any name appearing between "<" and ">" in any syntax diagram.

6. If the SEMANTICS option is used, then the semantics will be displayed instead of a syntax diagram.

7. Any <verb-name> or <syntax-item> must be spelled exactly (no abbreviations).

8. The file DMS/HELPINQ must be present.

NEXT

Option 1:

```
NEXT ----------------------------------------------------------------->#
-                     |                      |
                      |--- <data-set-id> --->|
```

Option 2:

```
NEXT -------------------- ? ------------------------------------------>#
-
```

Option 1

1. Continue the selection of records as previously specified.

2. If no <data-set-id> is given, continue the selection process of the most recent command at the point at which it stopped.

3. If <data-set-id> is given, e.g., X, continue the most recent command which selects X at the point where it selects a record from X. This need not be the point at which the command had stopped.

Option 2

1. NEXT? asks the system to display the name of the next data set to be selected if NEXT is entered.

OPTIONS

```
OPTIONS ------ PRINTER ---------------------------------------->#
-         |                                              |
          |--- TERMINAL -------------------------------->|
          |                                              |
          |--- COMMENTS -------------------------------->|
          |                |   |                      |
          |--- NULL ------>|   |--- = --- TRUE ------>|
          |                |           |           |
          |--- QUOTES ---->|           |- FALSE -->|
```

1. OPTIONS allows a user to control certain actions during his session.

2. TERMINAL causes output from DISPLAY statements to go to the user's terminal. This action remains in effect until PRINTER is set. TERMINAL is set by default during initialization of INQUIRY.

3. PRINTER diverts output from DISPLAY statements to a printer. This action remains in effect until TERMINAL is set.

4. Setting TERMINAL or PRINTER can be done at any time.

5. COMMENTS or COMMENTS = TRUE causes the quoted comments to be displayed along with identifiers when using the SHOW command. COMMENTS = FALSE suppresses the display of those comments. COMMENTS is FALSE by default.

6. NULL = FALSE will NOT display the names of items whose value is NULL when TERMINAL FORMAT TAB or SINGLE is used.

   NULL or NULL = TRUE (the default) displays all items regardless of value.

7. QUOTES or QUOTES = TRUE (the default) causes INQUIRY to require quote marks to bracket all alpha-literals.

   QUOTES = FALSE causes INQUIRY to recognize alphabetic or alpha-numeric strings as alpha-literals when associated with alpha variables.

PRINTER

```
PRINTER --------------------------------------------------------->|
-           |                                     |
            |<---------------- , -----------------|
            |                                     |
            |----- PAGE ------------ <integer> -->|
            |   |         | |     |               |
            |   |- WIDTH -| |- = -|               |
            |                                     |
            |- FORMAT ----------- HEADING ------->|
                       |     | |                  |
                       |- = -| |- TAB ----------->|
                               |                  |
                               |- SINGLE -------->|
```

1. PRINTER allows the user to display or alter printer attributes.

2. Entering PRINTER with no other option will result in a display of the current printer attributes.

3. PAGE is the number of lines per page.

4. WIDTH is the number of characters per line.

5. The FORMAT option controls the way data is formatted on the printed page. See OUTPUT FORMATTING for a discussion of the FORMAT attribute.

6. The printer attributes are active only if OPTION is PRINTER.

QUIT

```
QUIT ------------------------------------------------------------>#
-
```

1. QUIT terminates the inquiry of a database.

2. Used only when the user is finished. It is necessary to re-initiate DMS/INQUIRY if further inquiry of this or another database is desired.

3. If any text associated with a DEFINE, VIRTUAL, and/or GENERATE statement has been added, deleted or modified, entering QUIT will result in the warning:

   ```
   # DECLARATIONS NOT SAVED
   ```

   Entering QUIT (or a null input) will result in the termination of INQUIRY without the text being saved. To save the text, enter SAVE (see SAVE verb); INQUIRY will abort the termination process if anything other than QUIT, SAVE or a null input is entered at this point.

RECALL

```
RECALL ---- <data-set-id> --------------------------------------->#
---        |                                            |
           |--- <temp-set-id> ------------------------->|
           |                                            |
           |--- <virtual-id> -------------------------->|
           |                                            |
           |--- DEFINE --- <define-id> ---------------->|
```

1. This verb places the text associated with identifier into the current text buffer.

2. If <data-set-id> is specified, then the SELECT and/or DISPLAY text associated with the data set is placed into the current text buffer and displayed.

3. If <temp-set-id> is specified, the GENERATE text is placed into the current text buffer and displayed.

4. If <virtual-id> is specified, the VIRTUAL text is placed into the current text buffer and displayed.

5. If DEFINE <define-id> is specified, the text associated with the define-id is placed into the current text buffer and displayed.

6. In all cases, the RECALLed text can be EDITed and REPEATed.

REPEAT

```
REPEAT --------------------------------------------------------->/
-           |                       |
            |<-- <data-set-id> --->|
```

1. REPEAT by itself will reprocess the current text buffer (see SHOW and RECALL).

2. It is anticipated that the REPEAT verb will be used following an EDIT or RECALL statement.

3. REPEAT <data-set-id> will begin the selection process for the select and display specifications associated with the specified structure as well as SELECT and/or DISPLAY statements associated with attached, subordinate structures. Note that though the owner data set of <data-set-id> is attached, there is no automatic selection of the next record of the owner.

RESTORE

```
RESTORE -----+------------------------------------------------->#
--+      |                                   | |              |
         |                                   | |-- OVERWRITE ->|
         |---+-------------- <file-name> -->|
         |                  |                |
         |-------- * ------>|                |
         |                  |                |
         |- (<usercode>) ->|                 |
         |                                   |
         |-- <B1800-file-identifier> ------>|
```

1. RESTORE reloads the text from the file previously SAVEd. If the file does not exist, the user will be so informed.

2. If the option OVERWRITE is not used, id's that already exist will not be overwritten from the file.

3. If the option OVERWRITE is used, the text associated with existing id's which have the same name as id's on the file will be replaced from the file.

4. The verbs SHOW GENERATE, SHOW DEFINES, and SHOW VIRTUALS can be used to determine what is loaded.

5. To reestablish the contents of temporary sets, it will be necessary to RECALL and REPEAT the desired set id's.

6. The <file-name> is a 10-character file identifier.

7. The <B1800-file-identifier> is a full file identifier conforming to B1800 file identifier syntax, specified in the MCPII Product Specification.

8. If an asterisk (*) is specified, the file is assumed to reside on the system disk.

9. If a (usercode) is specified, the file is assumed to have that (usercode) as a multifile-id and is assumed to reside on the default pack for that usercode.

10. If the specified file does not exist, DMS/INQUIRY will give an error and continue processing.

SAVE

```
SAVE ---------------------------------------------------------------->#
--      |                                                          |
        |------------------------  <file-name> --->|
        |                         |                                |
        |------- * --------->|                                     |
        |                         |                                |
        |-- (<usercode>) -->|                                      |
        |                                                          |
        |-- <B1800-file-identifier> ------------->|
```

1. SAVE saves the text associated with DEFINE, VIRTUAL and GENERATE statements.

2. If * precedes the <file name>, the file saved becomes a system file.

3. If <(usercode)> precedes the file name, the file will be saved in the usercode directory.

4. If neither a * nor a <(usercode)> precedes the filename, the file will be saved in the terminal user's directory.

5. See RESTORE to recover the text SAVEd.

6. The use of the prefix * or <(usercode)> may cause the termination of INQUIRY without saving any text dependent upon the security clearance of the terminal user.

7. The <B1800-file-identifier> is a full file identifier conforming to B1800 file identifier syntax, as specified in the MCPII Product Specification.

SELECT/DISPLAY

```
------DISPLAY-----<display-spec>------------------------------------->#
 | |  -                          | |                               | |
 | |                             | |                               | |
 | |                             | |                               | |
 | |---SELECT---<select-spec>--->| |-- , --REPEAT--<data-set-id>-->| |
 |      -                                   -                        |
 |                                                                   |
 |<-------------------------- , -------------------------------------|
                 |       |      |
                 |       v      |
                 |<-THEN--------|
```

In the syntax diagram above, each <display-spec> or <select-spec> contains selection criteria for one structure; e.g., Xi, at nesting level i. Assume that the entire command contains selection criteria for several levels of embedded structures: Xn, Xn+1, . . ., Xm. The command specifies the current select/display information attached to data set Xi. If the command does not specify proper syntax then none of the data sets current information is changed.

The basic action of the command is to select one record from each structure Xi (n<=i<=m) which satisfies the selection condition for Xi. In addition, certain items are displayed on the user's terminal each time a record is selected by a DISPLAY clause. This action may be repeated automatically until all selected structures are exhausted, or, under certain conditions, the action will periodically stop and wait for further input (e.g., NEXT) from the user.

1. The DISPLAY and SELECT clauses must be arranged so that the structure Xi is embedded in Xi-1 for n < i <= m.

2. The limit, if present, is an integer constant >0.

3. All Xi, i<n, must have already been selected; i.e., there must be a current record for X1, X2, ..., Xn-1.

4. The selection criteria and display lists for all data sets of the tree for Xm (except Xm) are set to empty, and those structures are marked as having no current records.

5. The <select-spec> or <display-spec> for Xi is "attached" to structure Xi. That is, the given selection criteria and <display-list>s are remembered for subsequent use. Any previous criteria or lists are discarded.

6. Records are selected from the Xi in order of the nesting level. For each Xi selected by a DISPLAY clause, items are

displayed as soon as the record is selected. This process continues until one record from each Xi has been selected.

7. If Xm (the innermost structure selected) is not selected by a DISPLAY clause, the selection process will stop after selecting a record from Xm, and wait for further input.

8. If Xm is selected by a DISPLAY clause, the system will continue to select and display records of Xm.

9. If a limit is associated with any Xi, this value is tested prior to attempting to select a new record from Xi. If the limit is exhausted (zero), the system will stop and wait for further input. Otherwise, the limit is decremented and the selection process continues.

10. If, for any Xi, where i>n, no more records from Xi satisfy its selection criteria, the system will select a new record from Xi-1 and then continue the selection of Xi in the new Xi structure. Whenever Xn is exhausted, the entire process ends.

11. Whenever Xi is selected, the limit values for all structures embedded in Xi (Xi+1) are reset to their original values. Also all data sets of the tree structure of Xi (except Xi) are marked as having no current record.

12. Whenever the system is stopped and waiting for further input, the command NEXT Xi (for n<=i<=m) may be employed to cause the system to continue the selection process just described with the next Xi record which satisfies the selection criteria for Xi. Xi need not be the structure about to be selected when the system stopped. Exactly one record is selected from each structure Xi, Xi+1, ..., Xm.

13. The command NEXT is identical to NEXT Xi, where Xi was the next structure to be selected when the system stopped.

14. Note that whenever the system stops to wait for further input, a consistent set of records exists as current records, one from each structure X1, X2, ..., Xi, i<=m.

15. For purposes of automatic looping, note that the structures Xn, ... Xm may be considered part of a single set of nested loops (Xj is attached to Xj-1, n<j<=m). That is, whenever a record of xi, n<=i<m, is selected, records are automatically selected for all Xj, i<j<=m; and whenever any Xi, n<i<=m, is exhausted, a record is selected from structure Xi-1. Prior to this command, if Xn had already been specified, and was in fact attached to structure Xn-1, it is now detached from Xn-1. Also if Xm+1 had previously been attached to Xm, it is now detached from Xm (See ATTACH and DETACH).

16. The REPEAT clause is used to cause automatic looping outside

the normal family of embedded structures. If the REPEAT clause (e.g. REPEAT Z) occurs after selection criteria for, say, Xi, then the REPEAT clause is attached to structure Xi (Same as SET Xi to REPEAT Z). After a record of Xi is selected, Z is automatically REPEATed. All of its automatic selection and display will take place, just as if REPEAT Z were entered after the selection of a record of Xi. When the REPEAT Z action is completed, the processing of the original family (Xi) resumes in the normal manner. Note that if Z is an embedded data set that is attached to its owner, the automatic selection includes only attached data sets embedded in Z. The REPEAT may not require record selection from data sets at a higher nesting level of the branch X1...Xn..Xi...Xm where X1 is a disjoint data set.

SELECTION CRITERIA

<select-spec>

```
-------------- <condition> ---------------------------------->>
 | |    |                                                   |
 | |- AT ->|                                                |
 |                                                          |
 |----------- <structure-id> ------------------------------>|
   |      |                           |                    |
   |- VIA ->|                         |- AT -- <condition> ->|

>>------------------------------------------------------------>#
          |                                  |
          |---- , SORT ---- <sort-spec> ---->|
```

<display-spec>

Option 1:

```
-------------- <display-list> ----------------------------->>
  |        |                  |                      |
  |-<limit>->|                |- VIA -- <structure-id> -->|

>>----------- AT --- <condition> ------------------------------>#
                                 |                        |
                                 |-- , SORT -- <sort-spec> -->|
```

Option 2:

```
----------------------------- <structure-id> ------------------->>
  |        | |       |
  |-<limit>->| |- VIA ->|

>>------------------------------------------------------------------>#
          |                      | |                              |
          |- AT -- <condition> ->| |-- , SORT -- <sort-spec> -->|
```

1. "WHERE", "IN", "FOR" and "a" are synonyms for "AT".
2. "USING" is a synonym for "VIA".

1. A condition is a general boolean expression, involving <boolean primary>s combined by AND, OR, and Parenthesis (See SELECTION CONDITION for details).

2. Selection criteria may contain either or both of two components, the <structure-id> and the <condition>.

3. If a <structure-id> is specified, that structure is used to access records. The <structure-id> may be a data set, an automatic or manual subset, a temporary subset or a link. In all cases, the data set from which the records are ultimately read is the structure, (X), to which the selection criteria are attached. Records are accessed in the order of the given structure. If a <condition> is not given, all records are selected. The keyword VIA is optional in most cases; its use aids in the processing of the command, but it has no effect on the meaning of the command. If no structure-id is specified (then there must be a condition), records will be accessed by whichever structures the system determines to be the most efficient in satisfying the condition.

4. The condition specifies certain relationships which must be satisfied by items in a record if that record is to be selected (See SELECTION CONDITION).

5. The SORT OPTION allows records to be selected (and displayed) in the order of one or more keys. See SORT OPTION for details.

6. Limit is an unsigned integer greater than 0.

SELECTION CONDITION

<condition>

```
      |<--------------------- AND --------------------|
      |                                               |
      |<--------------------- OR  --------------------|
      |                                               |
>--------------------------------------------------------------->#
         |        |  |                            |
         |        |  |----- <boolean-primary> -->|
         |--NOT-->|  |                            |
                     |--- ( --<condition>-- ) -->|
```

<boolean-primary>

```
>-------- <aexp> ------------- <relop> ---- <aexp> ------------>#
 |                                                            |
 |                                                            |
 |-- <alpha-item> ----------- <relop> -- <alpha-literal> --->|
 |                                      |                  |  |
 |                                      |- <alpha-item> -->|  |
 |                                                            |
 |-- <boolean-function> ------------------------------------->|
```

where <relop> is

SPELLING	MEANING
EQL, =	(EQUAL)
NEQ, ¬=	(NOT EQUAL)
LSS, <	(LESS THAN)
LEQ, <=, ¬>	(LESS THAN OR EQUAL TO)
GTR, >	(GREATER)
GEQ, >=, ¬<	(GREATER OR EQUAL TO)

1. The <condition> is a generalized boolean expression consisting of boolean primaries ANDed/ORed together. The use of parenthesis allows the ANDing/ORing of conditions to form complex conditions.

2. The <boolean-primary> establishes a truth value dependent on the relationship between two values.

3. The <aexp> is a general arithmetic expression involving at least one arith-item (i.e., items of type NUMBER) of the record to be selected.

4. The <alpha-item-1> is an item (see <item>) of type ALPHA.

5. The object dataset of the selection expression will be determined according to the algorithm specified in Section 2.

6. Precedence of evaluation of logical and arithmetic operations is as specified in the DASDL standard.

DISPLAY-LIST

<display-list>

```
             |<------------- , -----------------|
             |                                  |
>-------------------- <item> ------------------------------------->#
                 |                          |
                 |-- <occurrence-segment> ->|
                 |                          |
                 |--------- <aexp> -------->|
```

<occurrence-segment>

```
-- <identi -------------------------------------------------------->#
    fier>  |                 |  |                                 |
           | |<---- , -----| |  |   |<---------- , -----------|   |
           | |             | |  |   |                         |   |
           |---OF <quali --->|  |- (--<sub --- TO --<sub -----) ->|
                   fier>                 script1>     script2>
```

1. A <display-list> specifies items to be displayed on the user's terminal.

2. All items must be contained in the record being selected or in data sets having a current record.

3. An <occurrence-segment> is a short-hand notation representing several occurrences of an occurring item, beginning with <subscript1> and ending with <subscript2>. Subscript1 and subscript2 must be unsigned integers. An occurring item with no subscripts is an <cccurrence-segment> specifying all occurrences of the item.

4. The term <item> includes virtual items, defined by the terminal user with a VIRTUAL statement. Virtual items may not be qualified (all have unique names) and they may not be subscripted.

SORT OPTION

<sort-spec>

```
                  |<---------------- , ------------------|
                  |                                      |
-------------------------- <key-item> --------- ASCENDING -------->#
    |         |                           |                   |
    |-- ON -->|                           |---- DESCENDING -->|
```

<key-item>

```
---- <alpha-item> ------------------------------------------------>#
  |               |
  |- <numeric-item> ->|
```

The keywords may be abbreviated where the minimum recognized abbreviation is the underlined part below:

```
    ON             ASCENDING
    --             ---

    DESCENDING     SORT
    ----           --
```

1. The SORT Option allows selecting or displaying records in order of a specified key.

2. The SORT Option is specified after the SELECT <condition> in a SELECT or DISPLAY statement.

3. The SORT Option may only be specified in a selection expression for a disjoint dataset. The use of the SORT Option will cause a SORT/¥SORT task to be generated to effect the sort.

Examples:

```
    DISPLAY D1,D2 AT D3 > 50, SORT ON D4
    SELECT D3 > 50, SORT ON D4, THEN DISPLAY E1,E2 AT E3 > 0
```

4. All <key-items> must be items in the selected data set. Qualification of <key-items> cannot be specified.

5. If a <key-item> is an occuring item, it must be subscripted by an integer constant.

6. The number of <key-items> specified cannot exceed 25.

7. The SORT Option associated with a data set can be established, modified or eliminated by use of a RECALL, EDIT and/or SET statement.

8. The SORT verb can be used to specify core resources to be used for sorting. The sort core specified will control the speed of the sort process.

The SORT Option functions are as follows:

1. Each record of the data set that meets the requirements of the SELECT-CONDITION is read from the data base. An entry is made in a tag file that consists of the extracted key-items and the data base address of the record.

2. The tag file is then sorted.

3. INQUIRY then reads the sorted tag file sequentially. For each tag entry, INQUIRY reads the data set record at the address indicated in the tag entry.

4. From this point, INQUIRY will perform as if the data set record was obtained directly from the data set, e.g. INQUIRY will display items or execute the statement associated with any attached embedded data set.

Caution: The user should be aware that, since all records must be selected, then sorted, before any display can occur, the terminal response may be considerably slower when using the SORT Option.

FUNCTIONS

<boolean function>

```
--- ANY ----- ( --------------------------- <condition> -- ) -->#
 |         |    |                         |
 |- ALL -->|    |- <structure --- WHERE ->|
                     -id>
```

1. ANY yields the truth value TRUE if any record in the dataset meets the requirements of the condition.

2. ALL yields the truth value TRUE only if all records in the dataset meet the requirements of the condition.

3. If the function is referenced at a given nesting level then the terms in <condition> must reference at least one item in a dataset at the next lower nesting level.

4. If a <structure-id> is specified, then only records retrieved via that structure are considered.

<arithmetic-function>

```
>--SUM---- ( --<aexp>--------------------------------------- ) -->#
 |       |           |                  | |                |   |
 |-AVG-->|           |                  | |                |   |
 |       |           |-VIA <structure ->| |-WHERE <condi ->|   |
 |-MAX-->|                     id>                tion>        |
 |       |                                                     |
 |-MIN-->|                                                     |
 |       |                                                     |
 |-SSQ-->|                                                     |
 |       |                                                     |
 |-MSQ-->|                                                     |
 |       |                                                     |
 |-VAR-->|                                                     |
 |       |                                                     |
 |-STD-->|                                                     |
 |                                                             |
 |                                                             |
 |-SCALE----- ( ---- <aexp> ----- , ---- <aexp> ----- ) ------>|
 |                                                             |
 |-ABS------- ( ---- <aexp> ----- ) -------------------------->|
 |       |                                                     |
 |-SQRT->|                                                     |
 |                                                             |
 |-COUNT---- ( ---<structure-id>------------------------- ) -->|
                |               |                       |
                |               |--WHERE--<condition>-->|
                |                                       |
                |--<condition>------------------------->|
```

1. For the above functions COUNT and SUM thru STD, if the function is referenced at a given nesting level (excluding the disjoint data set level) then the terms in <aexp> and <condition> must reference at least one item of the data set at the next lower nesting level. For all these arithmetic functions, if the WHERE condition is not specified then all records are utilized to produce the function value.

2. For the functions SCALE thru SQRT, the <aexp> may be any general arithmetic expression.

3. If a <structure-id> is specified, then only records retrieved via that structure are considered.

4. The functions are defined as follows:

   a. COUNT - The number of records which satisfy the <condition> (if specified), or the number of records in the specified structure.
   b. SUM - Sum of all <aexp>.
   c. AVG - Average = SUM/N.

d. MAX - Maximum, i.e., the value of the largest arithmetic item.
e. MIN - Minimum, i.e., the value of the smallest arithmetic item.
f. SSQ - Sum of squares.
g. MSQ - Mean square = SSQ/N.
h. VAR - Variance = (SSQ-N*AVG**2) / (N-1).
i. STD - Standard deviation = SQRT(VAR).
j. SCALE(X,F) - Return the value X scaled to F fractional digits by rounding. The non-fractional digits are unaffected.
k. ABS(X) - Absolute value.
l. SQRT(X) - Square root, X >= 0.

5. The arithmetic function is undefined if no records satisfy the condition.

ARITHMETIC EXPRESSIONS

<aexp>

```
                    |<--- <arith-op> -----------|
                    |                           |
>------------------------ <arith-primary> --------------------->#
       |         |     |                      |
       |--- - -->|     |                      |
                       |--- ( --<aexp>-- ) -->|
```

where <arith-op> can be

SYMBOL	MEANING
+	(ADD)
-	(SUBTRACT)
*	(MULTIPLY)
/	(DIVIDE)
DIV	(INTEGER DIVIDE)
MOD	(REMAINDER)

<arith-primary>

```
>-------------------- <numeric-constant> ----------------------->#
        |                                        |
        |----------- <arith-item> --------------->|
        |                                        |
        |----------- <arithmetic-function> ------>|
        |                                        |
        |----------- <virtual-item> ------------->|
```

1. The <aexp> is a generalized arithmetic expression. If used in a SELECT condition, all arithmetic items must be in data sets having current records or in the data set being selected. Parentheses can be used to combine arithmetic expressions into more complex arithmetic expressions.

2. A <numeric-constant> is a number.

3. An <arith-item> is an item in a record of a type which will yield a numeric value (See <item> below).

4. An <arithmetic-function> is specified in FUNCTIONS (See below).

5. A <virtual-item> is an item established by the VIRTUAL verb (See VIRTUAL).

ITEMS

<item>

```
- <identifier> ----------------------------------------------->#
               |                   | |                        |
               |--OF--<qualifier>->| |- ( -<subscript>--- ) ->|
                |                |         |            |
                |<---------------|         |<---- , ----|
```

1. An <identifier> is the name of a database item.

2. An <identifier> may be qualified, if necessary, by a <qualifier>.

3. A <qualifier> is the <identifier> of a data set which contains the <identifier>.

4. An <item> must be subscripted if it is defined with an occurs clause in DASDL.

5. A <subscript> is an arithmetic expression (see <aexp>) which must yield an integer value which does not exceed the DASDL defined OCCURS limit.

SET

```
SET -- <data-set-id> ------------------------------------------>#
---                     |                                      |
                        |                                      |
   |<-------------------|                                      |
   |                                                           |
   |---------+--- SELECT ---- <select-spec> ------------------>|
   |       | |           |                  |               |
   |       | |           |                  |               |
   |- TO ->| |           |---- NONE ------->|               |
             |                                              |
             |-- DISPLAY ---- <display-spec> -------------->|
             |            |                                 |
             |            |-------------  <display-list> -->|
             |            | |          |                    |
             |            | |-<limit>->|                    |
             |            |                                 |
             |            |------- NONE ------------------->|
             |                                              |
             |-- LIMIT ------------------ <limit> --------->|
             |          |       |     |             |       |
             |          |-- = ->|     |--- NONE  -->|       |
             |          |       |                           |
             |          |- TO ->|                           |
             |                                              |
             |-- REPEAT --- <data-set-id> ----------------->|
             |          |                 |                 |
             |          |--- NONE ------->|                 |
             |                                              |
             |                                              |
             |-- SORT --------- <sort-spec> --------------->|
                            |                  |
                            |---- NONE  ------>|
```

1. The SET statement allows the user to modify or remove the <selection-condition>, <display-list>, <repeat-spec>, <sort-spec> or <limit> previously attached to any structure.

2. The SELECT option attaches a new selection-condition to a structure, or removes a previous one (SELECT NONE). Any embedded data set attached to the structure,say Xi, is detached. If NONE, the given structure (Xi) will no longer be selected until a new selection condition is specified for Xi. Also, any structures subordinate to Xi (Xj where j>i) will no longer be selected.

3. The DISPLAY option attaches a new display-list to a structure, including a new selection condition if desired, or removes the previous display-list (DISPLAY NONE). The

current limit will be changed only if the display-spec specifies a new limit. Similarly, the selection criteria (VIA structure and/or AT condition) will be changed only if the display-spec specifies new selection criteria.

4. The LIMIT option attaches a new limit to a structure, or removes a previous one (LIMIT NONE).

5. The REPEAT option attaches a new repeat-spec to a structure, or removes a previous one (REPEAT NONE).

6. The SORT option attaches a new <sort-spec> to a structure, or removes a previous one (SORT NONE).

7. The SET statement does not, by itself, select any new records or display any items. However, any structure whose selection-condition is changed or removed is marked as having no current record selected.

8. The meaning of a subsequent NEXT (giving no structure-id) may be affected by changing the selection condition of a structure. If NEXT means NEXT Xj, j>i, then:

   a. After SET Xi TO SELECT <select-spec>, NEXT behaves like REPEAT Xi;

   b. After SET Xi TO SELECT NONE, NEXT behaves like NEXT Xi-1.

Note: For syntax of <select-spec> and <display-spec> see SELECTION-CRITERIA. For syntax of <sort-spec> see SORT-OPTION.

SHOW

```
SHOW ----------------------------------------------------------->#
--        |                                         |
          |--- ALL -------------------------------->|
          |                                         |
          |--- DATASETS --------------------------->|
          |                                         |
          |--- DEFINES ---------------------------->|
          |             |           |               |
          |             |-- <id> -->|               |
          |                                         |
          |--- VIRTUALS --------------------------->|
          |                                         |
          |--- GENERATES -------------------------->|
          |                                         |
          |--- DECLARATIONS ----------------------->|
          |                                         |
          |--- <identifier> ----------------------->|
            |                    |            | |
            |                    |--- SETS -->| |
            |                    |            | |
            |                    |--- ITEMS ->| |
            |                                   |
            |<------------------ , -------------|
```

1. SHOW, by itself, will display the text in the current text buffer.

2. SHOW ALL will display the DASDL description of the accessible data base.

3. SHOW DATASETS will display the names of all data sets in the accessible data base.

4. SHOW DEFINES will display the names and text associated with all defines. If the <id> option is used, the text for only that id will be displayed.

5. SHOW VIRTUALS will display the names and text associated with all virtual items.

6. SHOW GENERATES will display the names and text associated with all temporary sets.

7. SHOW DECLARATIONS will display the names and text associated with all GENERATE, VIRTUAL and DEFINE identifiers.

8. SHOW <identifier> will:

   a. If the <identifier> is the name of an accessible data set, then the system will display:

1. All sets and subsets of the data set including the names of keys and data in key items.

2. The names and description of all items in the data set.

b. If the <identifier> is the name of a set, then the owner of the set as well as the keys and data in key items will be displayed.

c. If the <identifier> is the name of a data base item then the description of all items by that name will be displayed as well as the owner data set of each item.

9. SHOW <identifier> SETS will display the sets of the identifier, which must be a data set name.

10. SHOW <identifier> ITEMS will display the names and description of all items of the identifier, which must be a data set.

SORT

```
SORT ---------------------------------------------------------->#
--          |                                            |
            | |<------------------- , ----------------|  |
            | |                                       |  |
            |------ CORE ----------------- <integer> --->|
                               |        |
                               |-- = -->|
```

1. Entering SORT with no other option will display the current values for SORT CORE.

2. SORT is used to specify the system resources to be used when called for by the SORT Option of the DISPLAY verb.

3. The value of SORT CORE defaults to the SORT/VSORT default of 8,000 bytes.

TERMINAL

```
TERMINAL ----------------------------------------------------->#
-         |                                                 |
          |<--------------------- , ------------------------|
          |                                                 |
          |--- PAGE ------------------------- <integer> --->|
          |                 |   |        |                  |
          |--- WIDTH ------>|   |-- = -->|                  |
          |                                                 |
          |--- SCREEN ---------------- TRUE --------------->|
          |            |        |  |           |            |
          |            |-- = -->|  |-- FALSE ->|            |
          |                                                 |
          |--- FORMAT ---------------- HEADING ------------>|
                       |        |  |            |
                       |-- = -->|  |-- TAB ---->|
                                   |            |
                                   |-- SINGLE ->|
```

1. TERMINAL allows the user to display or alter terminal attributes.

2. Entering TERMINAL with no other options will result in displaying the current terminal settings.

3. SCREEN is TRUE for CRT devices, otherwise it is FALSE.

4. PAGE is the number of lines per screen for CRT devices.

5. WIDTH is the number of characters per line. Some terminals are designed such that printing in the last character position of the line will cause an automatic line advance. The effects of this is that consecutive output will appear to be double spaced. This can be avoided by setting LINE to one less character position.

6. The system sets the values for SCREEN, PAGE and LINE to those of the terminal when DMS/INQUIRY is started.

7. The FORMAT option controls the way data is formatted on the users terminal. See OUTPUT FORMATTING for a discussion of the FORMAT attributes.

8. These terminal attributes are active only if OPTION is TERMINAL.

VIRTUAL

```
VIRTUAL --- <virtual-id> -------------------- = --- <aexp> -->#
-
```

1. The VIRTUAL statement establishes the given id as a virtual item. It has the value of the given arithmetic expression.

2. The virtual item can be used in either the display-item-list or in a selection condition.

3. The arithmetic primarys appearing in <aexp> must be such that when the virtual is referenced in a display list or selection condition, the items in <aexp> must either be in the record being selected or in data sets having current records.

4. See SELECTION-CONDITION for syntax of <aexp>.

5. The arithmetic expression for an established virtual may be changed only by

   a. CLEAR VIRTUAL and re-entering, or

   b. RECALL VIRTUAL, EDITing and REPEATing.

APPENDIX A - USE OF THE INQUIRY SYSTEM

OVERALL STRUCTURE

DMS INQUIRY consists of three basic components:

1. DMS/BUILDINQ, the BUILD INQUIRY program, which must be run first to set up necessary information in the INQUIRY CONTROL file, <database-name>/INQCTL. It will take designation of a logical database on the system as input, along with an optional set of valid system usercodes qualified to view the database. In addition, the program will accept input describing various parameters to enable the INQUIRY program to run more efficiently. Using the database dictionary created by the DASDL compile for the physical database containing the logical database, BUILD INQUIRY will create the INQUIRY CONTROL file. The file contains necessary information for control of the INQUIRY program and is associated only with this logical database.

2. DMS/INQUIRY, the second component of the system, is the INQUIRY program itself. A copy of this program will exist for every user running INQUIRY on the system. The program will ask the user for the name of the database which the user wants to inquire against and ensure that he is a valid user of that database. Reading the INQUIRY CONTROL file to get a description of the database, the program will initialize its internal tables and begin processing against the database. All INQUIRY processing is done by this program, and all INQUIRY code is shareable.

3. DMS/HELPINQ, the third component, is a data file which must be present in order for the HELP verb to function properly.

TERMINAL TYPES

The INQUIRY system will run on any terminal configuration which supports CANDE and SMCS (i.e., TD820, TD830, TTY).

RUNNING ENVIRONMENTS

Both DMS/INQUIRY and DMS/BUILDINQ may be run alone or under a message control system. When run alone (i.e., with only a datacomm handler) the program must be executed from the SPO and the internal file "REMOTE" must be equated to the file name of the station on which the program will be run. For example, if DMS/BUILDINQ is to be run on station 14 having a remote file name of S14, the following should be entered from the SPO:

```
?EX DMS/BUILDINQ FILE REMOTE NAME S14;
```

When running under an MCS such as SMCS, no file equation is required. The user enters:

```
     EX DMS/BUILDINQ          or, in the case of SMCS:

     SIGN ON DMS/BUILDINQ
```

Invocation of security will cause a change in the manner of execution. See the section labelled SECURITY.

BUILD INQUIRY (DMS/BUILDINQ)

The BUILD INQUIRY program may be run from a terminal or as a batch job taking input from a card file. The source of input is controlled by program switch 0. If SW0=0 terminal input is expected; otherwise, a card input file (labelled CARDS) is expected.

When taking input from a terminal, the program will initiate a dialogue to determine what functions it is to perform. With the user running under SMCS, the dialogue is as follows:

1. User enters "SIGN ON DMS/BUILDINQ"

2. The program responds with:

   "BUILD INQUIRY   VERSION X.X   MM/DD/YY"

   and asks:

   "DATA BASE NAME?"

   Legal responses are:

```
>--- <database name> ---------------------------------------->#
                              |                          |
                              |--- ON --- <pack-id> --->|
```

   If the <database name> does not exist, or the syntax of the expression is incorrect, the program will respond with an error message and recommence the dialogue. If a null input is entered, the BUILDINQ program will terminate.

3. Given an existing physical database, the program will query:

   "TOTAL OR LOGICAL DATABASE?"

   Legal responses are:

```
>------------------------------------------------------------>#
               |                |
               |--- LOGICAL --->|
               |                |
               |--- TOTAL ----->|
```

   If the user specifies "LOGICAL", the system will respond with:

   "LOGICAL DATABASE NAME?"

   The user must respond with a logical database existing within this physical database. Note that when a logical database name is specified, the INQUIRY CONTROL file will be generated with a multi-file id containing the logical (not physical) database name. If the user specifies "TOTAL", the entire database will be used.

If a null input is entered, the dialogue is aborted (without creating a control file) and recommences at step 2.

4. After determining which database is to be accessed, the program will ask:

   "PACK FOR INQUIRY CONTROL FILE?"

   If the INQCTL file is to reside on a pack other than the system pack, that pack-id should be given. A null input indicates that the control file will reside on the same pack as the dictionary.

5. The next question is:

   "TIMEOUT TIME?"

   DMS/INQUIRY will log off a user whose terminal remains inactive for a certain length of time. This gives the DBA a chance to specify a default for all inquiry users of this logical database. Response should be time in minutes, greater than 0 and less than 999. A null input assumes default, which is 15 minutes.

6. The user now must specify whether or not access is to be restricted to the inquiry programs. The question

   "SECURITY? (Y/N)"

   should be answered with a "Y" if restricted access to certain usercodes is desired (see SECURITY).

   If security is desired, the user then must supply the usercodes which are to be allowed to run DMS/INQUIRY against this database. This is specified by:

```
                 |<----------- , ---------------|
                 |                              |
>---------------- <valid system usercode> ---------------->#
```

The control file "<dbname>/INQCTL" is built from this information, and the dialogue loops back to step 2, where a new database may be specified. In this way, any number of inquiry control files can be set up during one terminal session.

To run from cards, the following deck must be used:

```
   ?EX DMS/BUILDINQ
   ?DA CARDS
    <card input>
   ?END
```

Card input will be free-format, one input per card, and must conform to the following diagram:

```
DATABASE ---  =  --- <physical  ------------------------- ; -->>
                    database name>  |                    |
                                    |-- ON <pack-id> -->|


>>- LOGICAL -----------------  =  -- <logical database -- ; -->>
            |                |                 name>
            |-- DATABASE -->|


>>- PACK --------  =  -------- <pack-id> ---------------- ; -->>


>>- TIMEOUT -----  =  -------- <integer> ---------------- ; -->>


                             |<----- , -------|
                             |                |
>>- USERCODES ----  =  ---------- <usercode> ------------ ; --->#
```

SECURITY

Security in the form of usercode checking is available in the INQUIRY system. This checking can be invoked when the BUILD INQUIRY program is run (see above). When invoking security, the DMS/BUILDINQ program should be run under a public, privileged usercode. The control file is created read only, without a usercode. Any user wishing to use the INQUIRY system through this control file must have a legal usercode associated with execution of the DMS/INQUIRY program.

TIMEOUT

If a user's terminal remains inactive for a specified length of time (specified in BUILD/INQUIRY), INQUIRY will terminate that user and go to EOJ. If DEFINEs, VIRTUALs, or GENERATEs exist which have not been saved, they will be saved in a file named "TIMEOUTXXX", where XXX is the LSN of the terminal that the user was running on.

DMS INQUIRY ("DMS/INQUIRY")

After creating an inquiry control file with DMS/BUILDINQ, the user may run DMS/INQUIRY against the database. DMS/INQUIRY will be able to run on any system which will support CANDE, and will run on any terminal which CANDE services. DMS/INQUIRY may be run either through a Message Control System (MCS) or alone. If it is run alone, the program must be initiated from the SPO, and file equated to the inquiry remote file "REMOTE" to the desired station. For example, to run on station #14 with a logical name of S14, enter:

```
   ?EX DMS/INQUIRY FI REMOTE NAME S14;
```

If run under an MCS, (such as SMCS), no file equation is required. Simply enter:

```
   ?EX DMS/INQUIRY
```

Once the program is running, it will print a header message notifying the user of the version and compile date of the program being used:

```
   B1800/B1700 DMS/INQUIRY VERSION X.X (MM/DD/YY)
   WHAT DATABASE?
```

The user replies with the name of the database he is inquiring into (if the database name contains any non-alphabetic characters, it must be enclosed in quotes). If a control file for the database exists, a check of his usercode is made (if security has been specified at BUILDINQ time). If the usercode is valid (specified at BUILDINQ time), access is granted. Otherwise, access is refused. The reply must be formatted as follows:

```
--- <database name> ---------------------------------------------->#
                          |                         |
                          |--- ON --- <pack-id> --->|
```

If access is granted, the program responds with:

```
   INITIALIZING...PLEASE WAIT...
```

The user is notified when initialization is complete by the message:

```
   ...READY
   #
```

and processing may begin.

TESTING FEATURE

In order to ease testing of DMS/INQUIRY, a feature has been added to enable input to be entered on cards exactly as it would be entered through the terminal. This file is called "COMMANDS" and is assumed to be a serial file containing DMS/INQUIRY instructions. This capability is controlled by program switch 9. To use this feature:

```
   EX DMS/INQUIRY FI COMMANDS NAME CARDS; SW9=1;
```

A card file named "CARDS" will be expected as primary input. Output can be directed to disk by setting the PRINTER option in DMS/INQUIRY and file equating the file "LINE" to a disk file at execution time.

NOTE: This feature is intended for use only by the Santa Barbara facility.

PERFORMANCE CHARACTERISTICS

Any performance prediction is, at best, a risky affair. While this design has attempted to minimize data and code space, as well as execution time, performance of the inquiry system could vary dramatically depending on many factors; e.g., memory capacity of the system, the number of concurrently running jobs, the design and size of the database, and the DMS INQUIRY constructs used to obtain the desired information.

This design also attempts to reduce the amount of static memory required to run the program. By placing most data in dynamic memory, the user is allowed a certain amount of control over run-time memory requirements not otherwise possible. Execution-time specification of dynamic memory space requirements allows the inquiry program to utilize as much memory as possible in any situation. This should have a positive effect on the performance of the system.

The design of the database could have a profound effect on the performance of DMS INQUIRY. Utilization of existing structures in the database will, in general, result in much better performance than use of DMS INQUIRY capabilities which do not take advantage of the database structure. The size of the database, both in population and in number of items, will also affect performance. The former increases the time used by the MCP to retrieve records for the inquiry process, while the latter increases the size and complexity of the inquiry program's internal database description.

Thirdly, the constructs used for inquiry purposes will have an effect on performance. Some constructs, such as SORT, require that the entire data set be looked at and sorted before any response appears to the user. Conversely, simple incuiries using existing index sets will result in better response. Additionally, overenthusiastic use of virtual items, temporary sets, and defines will have a negative impact on processing speed. In general, the simpler an inquiry is, the more satisfactory the performance will be.

In summary, optimal performance would probably be obtained under the following conditions:

1. Maximum dynamic memory space should be allocated for the inquiry program.

2. The object logical database should contain no more datasets than required.

3. Inquiries should be simply constructed, and should follow existing index sets.

4. Obviously, other programs in the mix could have a negative effect on the inquiry program's performance.

INDEX