QUERY/V
Reference Manual

HP 3000 MPE/iX Computer Systems

HP Part No. 30000-90042
Printed in U.S.A. OCT 98

E1098

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the data indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

First Edition	Jun 1976	32216A.00.00
Update 1	Apr 1978	32216A.03.05
Update 2	May 1979	32216A.04.01
Update 2 Incorporated	May 1979	32216A.04.01
Second Edition	Nov 1981	32216B.00.00
Third Edition	Dec 1985	32216C.00.00
Update 1	Apr 1986	32216C.00.00
Update 1 Incorporated	May 1986	32216C.00.00
Fourth Edition	May 1987	32216C.00.08
Update	Oct 1998	Converted to SGML format

PREFACE

This manual describes QUERY/V, the Hewlett-Packard Data Base Inquiry Facility which is designed to run under the control of the HP 3000 Multiprogramming Executive (MPE) Operating System. QUERY/V enables you to access data in IMAGE data bases without writing a computer program.

In order to use QUERY, you should know enough about the MPE Operating System to start a job or operate in session mode. It is recommended that you read an introductory book to the HP 3000 systems if you are not an experienced MPE user. Knowledge of EDIT/3000 is useful for some QUERY techniques but is not required to operate QUERY.

This manual documents QUERY versions B.01.01 and later. Information on Native Language Support (NLS) and the IMAGE data types I4, J4, and K2 has been added to this edition. NLS is supported on QUERY versions B.01.00 and later, and the IMAGE data types I4, J4, and K2 are supported on QUERY versions C.00.00 and later. Information which is specific to a particular version of QUERY (B.01.01 or later) is noted throughout this manual.

This manual is arranged in three sections and seven appendices. Section 1 introduces QUERY terminology, briefly describes IMAGE data bases, and discusses other important general IMAGE, QUERY, and MPE concepts. Section 2 illustrates operating QUERY in session or job mode and discusses the functions of QUERY and the commands which are used

to perform them. Appendix A contains all QUERY messages. Appendix B lists helpful information on numeric limits of QUERY commands and discusses the files used by QUERY during execution. Appendix C describes three ways to access a data base on a remote HP 3000. Appendix D discusses Native Language Support. Appendix E discusses self-describing files. Appendix F discusses user-defined procedures and gives examples in programming languages. Appendix G lists the ASCII character set.

In addition to this manual, you may need to consult the following manuals:

- *TurboIMAGE/XL Database Management System Reference Manual**
- *IMAGE/SQL Adminstration Guide**
- *MPE/iX Commands Reference Manual*
- *Using NS3000/iX Network Services*
- *Native Language Programmer's Guide*

Throughout this manual, no distinction is made between TurboIMAGE/XL and IMAGE/SQL. These manuals are both referenced as the *IMAGE Reference Manual*.

The QUERY product is now referred to as QUERY/V. In previous editions of this manual, the QUERY product was called QUERY/3000. This edition of the QUERY/V Reference Manual documents the C.00.08 version of QUERY.

Contents

Figures

Tables

1

INTRODUCING QUERY/V

QUERY/V provides a simple method of accessing IMAGE data bases without programming effort. You can use QUERY to do the following:

- enter data
- modify or delete data values
- retrieve data which meets selection criteria
- report on the data retrieved

You perform these operations by entering simple commands consisting of English keywords such as FIND and REPORT. You can request information about the function, format, and parameters of QUERY commands with the HELP command. All of the QUERY commands are discussed in Section 3.

You only need to know the relationships of the data base elements and not the structure of the disc files. You can request information about the structure of the data bases with the FORM command. QUERY finds the data and performs the operations in response to your commands using the data base, data set, and data item names you specify.

QUERY can be used either in session (interactive) mode or job (batch) mode. If you are using QUERY interactively at a terminal, QUERY prompts you for commands, issues error messages when an error occurs, and prints other information of interest to you. You can operate QUERY in job mode as an MPE job from an input device. In job mode, any error messages are printed on the $STDLIST device which is usually a line printer.

QUERY adheres to all of the IMAGE security provisions described in the *IMAGE Reference Manual*. You must enter a valid password for the data base you want to use. This password determines which information you can access in the data base.

QUERY report formatting capabilities enable you to design reports with header and column labels, page numbers, and group labels. In addition, you can sort entries through multiple fields, as well as total, average, or count columns of numeric data values.

A sequence of commands can be stored in a file and executed at any time by entering the XEQ command as part of a session or a job. A frequently used or complex command can be stored as an individual procedure in a file referred to as a Proc-file. The procedure name can then be used in place of the command parameters.

If your local HP 3000 and a remote HP 3000 have Distributed Systems (DS/3000) capability, you can use QUERY to access IMAGE data bases residing on the remote computer and retrieve information to be used on the local system. Three methods of accessing a remote data base are described in Appendix C.

QUERY allows you to write your own user-defined procedures in any programming language which enable a report to perform a specialized function not provided by QUERY. Refer to

Appendix F for further discussion. All of the tasks except user-defined report procedures can be accomplished without programming.

If you are developing programs which access IMAGE data bases through the IMAGE library procedures, QUERY makes an excellent debugging aid. You can alter the data base content using your program and then use QUERY to examine the data and determine the results of your programmed changes.

STRUCTURE OF THE DATA BASE

It is not necessary to understand all the IMAGE features in order to use QUERY. However, it is important to have a general idea of data base structure and know the definitions of some IMAGE terms which will be used in this manual.

Each item of information in a data base is referenced by a *data item name.* The name associates the information with characteristics which describe it:

- the type of information (numeric, alphanumeric, etc.)
- its relation to other data in the data base
- the passwords required to read and/or write the information

The data base designer organizes data items into *data sets* for the purpose of accessing them as a group. For example, an employee data set could contain the items EMPID, F-NAME, L-NAME, SOCSEC#, and SALARY. A credit union data set could contain EMPID, AMOUNT, TRANSCODE, and so forth. Each data set is referenced by a *data set name.*

Each time you enter a new employee's record into the data base, you can supply a value for each data item. This group of values is stored as a single *data entry* in the data set. For example:

```
F-NAME              SALLY

L-NAME              MERTON

SOCSEC#             527-58-6492

SALARY              18000.02
```

QUERY allows you to locate particular entries which have values you specify. You can then change the values or print them in a report. You can also add or delete an entire entry if your password gives you the capability to write to each item in the data set.

You can use the FORM command to display the names of each data set to which you have access and the names of the data items in those sets. Only the items to which you have at least read access are listed.

Fully-Qualified Data Item Names

If you are referring to a data item which appears in more than one data set and/or data base, you must specify which data set and data base to access. A particular data item can appear in more than one data set within a given data base, and a particular data item can appear in more than one data base. You can specify the data item in three ways:

- by specifying the data set through the DATA-SET= command and the data base through the DBLIST= command.
- by specifying the data set through the DEFINE command (DATA-SETS= prompt) or through the MULTIDB command (DATA-SETS= prompt).
- by qualifying the data item name.

A fully-qualified data item name has the following form:

data base name:data set name.data item name

For example:

BADGE# is the name of a data item in the data set named LABOR which is in a data base named IRONCO. If BADGE# is also the name of a data item in a data set named EMPLOYEE, which is also in the IRONCO data base, its fully-qualified name would be IRONCO:EMPLOYEE.BADGE#.

Data Types

The data base designer defines each data item as a particular type, depending on what kind of information is to be stored in the item. A data item may be one of several types of integers, real or floating-point numbers, or ASCII character information.

The FORM command can display the data type for each item. When using QUERY, you will usually be unconcerned with the specific data type with the following exceptions:

- when supplying values for an item, either to enter new information or to locate specific entries, you may want to know the acceptable range of values for a numeric type item.
- when creating reports you should be aware of the item types in order to format the report properly.
- when using the QUERY registers while printing a report, it is helpful to know how calculations affect the register values.

Detailed information about each of these situations is given with the appropriate command in Section 3. Table 1-1 contains a summary of the data item types and the range of acceptable values for each.

Data Values

If you use QUERY to enter a value for a data item of type P (packed decimal), you should be aware of the way QUERY handles the sign of the value. A different code is used for the sign of a value entered with a plus sign than for a value entered without a sign. However, when QUERY retrieves unsigned and positive type P data items with the same value, they are considered to be equivalent. For example, +2 and 2 are equivalent. Data items with values +0, 0, and -0 are also equivalent.

Literals

When specifying the value of a particular data item, you must sometimes surround the value with quotation marks. This type of value is called a literal. A character or string literal contains alphanumeric characters. For example:

```
"TANYA OAKLEY"          "ZXR=93458273"          "3215"
```

Character literals containing numeric values of the types listed under Integer or Real in Table 1-1 are called, more specifically, numeric literals. For example:

```
"5468"                  "+408E-15"              "-16.73892"
```

Rules for using quotation marks are described with the commands which allow or require their use.

Table 1-1. Data Item Types and Values

TYPE	MINIMUM	MAXIMUM
INTEGER		
I1	-32768	+32767
I2	-2,147,483,648	+2,147,483,647
I4	+9,223,372,036,845,775,808	+9,223,372,036,854,775,807
J1	-9999	+9999
J2	-999999999	+999999999
J4	-999999999999999999	+999999999999999999
K1	0	+65535
K2	0	+4,294,967,295
Z*n* *	- (*n* digit number)	+ (*n* digit number)
P*n* *	- (*n* - 1 digit number)"	+ (*n* - 1 digit number)"
REAL		
R2	-1.157920 x 10^{77} -0.863617 x 10^{-77}	+0.863617 x 10^{-77} +1.157920 x 10^{77} Largest accurate absolute integer is 8,388,607. (QUERY rounds to 6 digits when printing R2 values.)
R4	-1.157920892373161 x 10^{77} -0.8636168555094445 x 10^{-77}	+0.8636168555094445 x 10^{-77} +1.157920892373161 x 10^{77} Largest accurate absolute integer is 36,028,797,018,963,967. (QUERY rounds to 16 digits when printing R4 values.)
CHARACTER		
U*n* **	1 ASCII character (lower case not allowed)	*n* ASCII characters (lower case not allowed)
X*n* **	1 ASCII character"	*n* ASCII characters

Notes on Table 1-1.

* *n* cannot exceed 255 and must be even for type Z and evenly divisible by 4 for type P*n*. QUERY reports print at most 20 digits for type Z and 19 digits for type P data values.

***n* cannot exceed 255. QUERY reports print at most 136 characters for type U and X data values.

Compound Data Items

IMAGE allows the data base designer to specify compound data items. A compound data item is one that occurs more than once in the same data entry. Each occurrence of the data item is called a sub-item. Each sub-item can have a value, and QUERY can locate and update any or all sub-items. If you update only the first sub-item, QUERY preserves the existing values of all other sub-items.

Data Set Relations

There are three types of IMAGE data sets: *manual master, automatic master,* and *detail.* Master data sets are related to detail data set through specific items called search (or key) items. The FORM command identifies the data set type and search items.

The data base designer can specify one or more sort items. These items are also identified by the FORM command. As a QUERY user, it is not necessary to understand the function of sort items. If you want to know more about data set relations and sort items, refer to the *IMAGE Reference Manual* description of the data base structure.

Sample Data Base

Figure 1-1 and Figure 1-2 illustrate a sample data base named ORDERS. The data base is used in many examples in this manual. It contains six data sets. The four master data sets are shown in the center column and the detail data sets on the sides.

- CUSTOMER contains information about each customer.
- SUP-MASTER contains information about each supplier.
- PRODUCT contains information about each product.
- DATE-MASTER is an index of dates and can be used to retrieve information by date from the SALES or INVENTORY data sets.
- SALES contains credit and purchase information.
- INVENTORY contains product supply information.

Both figures show a single entry for each data set. Figure 1-1 contains the data item names, and Figure 1-2 contains a sample of the values in one entry. The arrows in both figures illustrate the relationship of the data sets through search items.

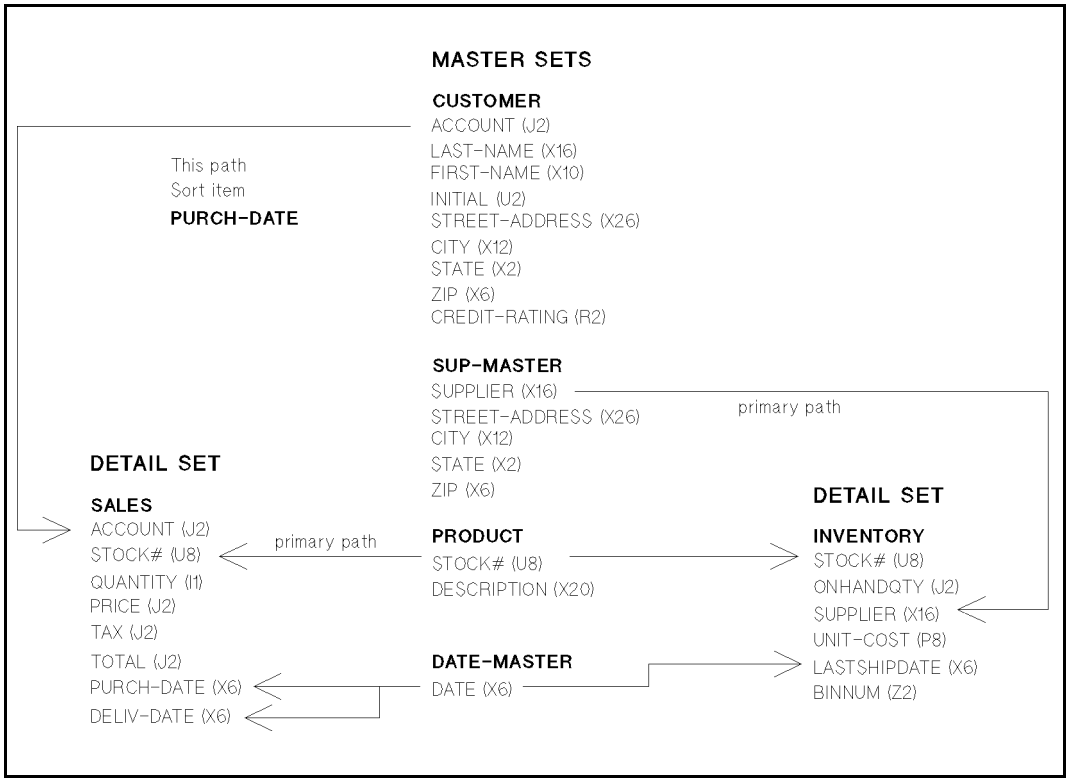


Figure 1-1. ORDERS Data Base Structure

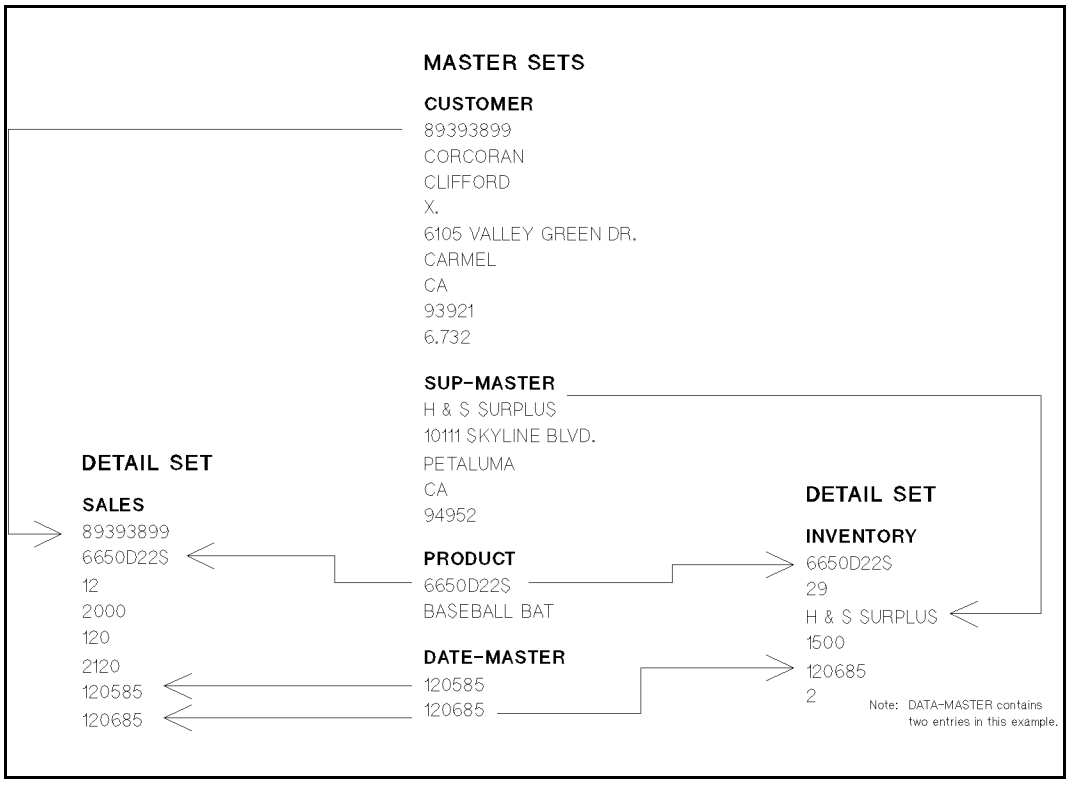


Figure 1-2. Sample Entry Values

Note: This data base is not meant to be a practical application but rather is designed to illustrate as many IMAGE/QUERY features as possible.

MODES OF ACCESS

Each person using QUERY to access a data base must specify one of the 8 modes of access. The following table lists the capabilities you will have in each mode.

MODE	CAPABILITIES
1	Find (read), replace, add, delete entries (QUERY requests IMAGE dynamically lock and unlock the data base when accessing it)
2	Find and replace entries
3* or 4	Find, replace, add, and delete entries
5	Find entries (QUERY locks and unlocks)
6, 7*, or 8	Find entries

* Modes 3 and 7 give you exclusive access to the data base. All other modes allow others to share the data base.

A data base can only be shared in certain situations or environments. The mode you specify must be acceptable for the environment already established by other IMAGE and QUERY users (if any) when you open the data base. Here is a summary of the acceptable environments:

- multiple mode 1 and mode 5 users
- multiple mode 6 and mode 2 users
- multiple mode 6 users and one mode 4 user
- multiple mode 6 and mode 8 users
- one mode 3 user
- one mode 7 user

Subsets of these environments are also allowed. For example, there may be all mode 6 users or all mode 8 users. There may also be one mode 1 user or all mode 5 users and so forth.

If a mode 3 or mode 7 user is currently accessing the data base that you want to access, you must wait until that user either terminates their IMAGE or QUERY session or accesses a different data base. This is true any time you try to access a data base with a mode which is incompatible with other users accessing the same data base. Changing your mode of access may enable you to access the data base.

When deciding which mode to use, you should consider the following:

- If you merely want to find information and examine or report on it, you should open the data base with a find (read-only) mode, thus allowing other users as much capability as possible. The following table gives the result of using each read-only mode.

MODE	RESULT
5	The data base is locked for some operations and may slow the rate of activity somewhat. If mode 1 or mode 5 users are already accessing the data base, you must use mode 5.
6	Mode 2 users can replace entries, and one mode 4 user can replace, add, or delete; or mode 8 users can read entries.
7	You have exclusive access to the data base.
8	No other users can replace, add, or delete entries.

An important advantage of using modes 5 through 8 is that the data base is is opened for reading only. As a result, you are more likely to gain access to the data base by avoiding restrictions due to the MPE account structure. Also, the files are not marked for inclusion in the MPE system backup tapes (SYSDUMP) since they are not altered in any way. This saves time when the daily system backup procedure is executed.

- If you want to find information and replace data in existing entries but do not need to add or delete any entries (and do not want anyone else to add or delete entries), you should open the data base with mode 2.
- If you want to perform all the operations, including adding and deleting entries, you should open with mode 1, 3, or 4. The following table gives the result of using each of these modes.

MODE	RESULT
1	QUERY locks and unlocks the data base while performing operations. Other users are able to add and delete entries. (The previous comments for mode 5 apply here also.)
3	You have exclusive access to the data base.
4	You have exclusive ability to change the data base but mode 6 users are able to read while you make changes.

Figure 1-3 illustrates acceptable IMAGE/QUERY environments. Users at terminals can operate IMAGE or QUERY simultaneously. A single user indicates that only one user may specify that mode. Two users indicates that multiple users may specify that mode. Terminals indicate the modes which are incompatible with the current environment. The capability of each user is abbreviated as follows: F = find, R = replace, A = add, D = delete.

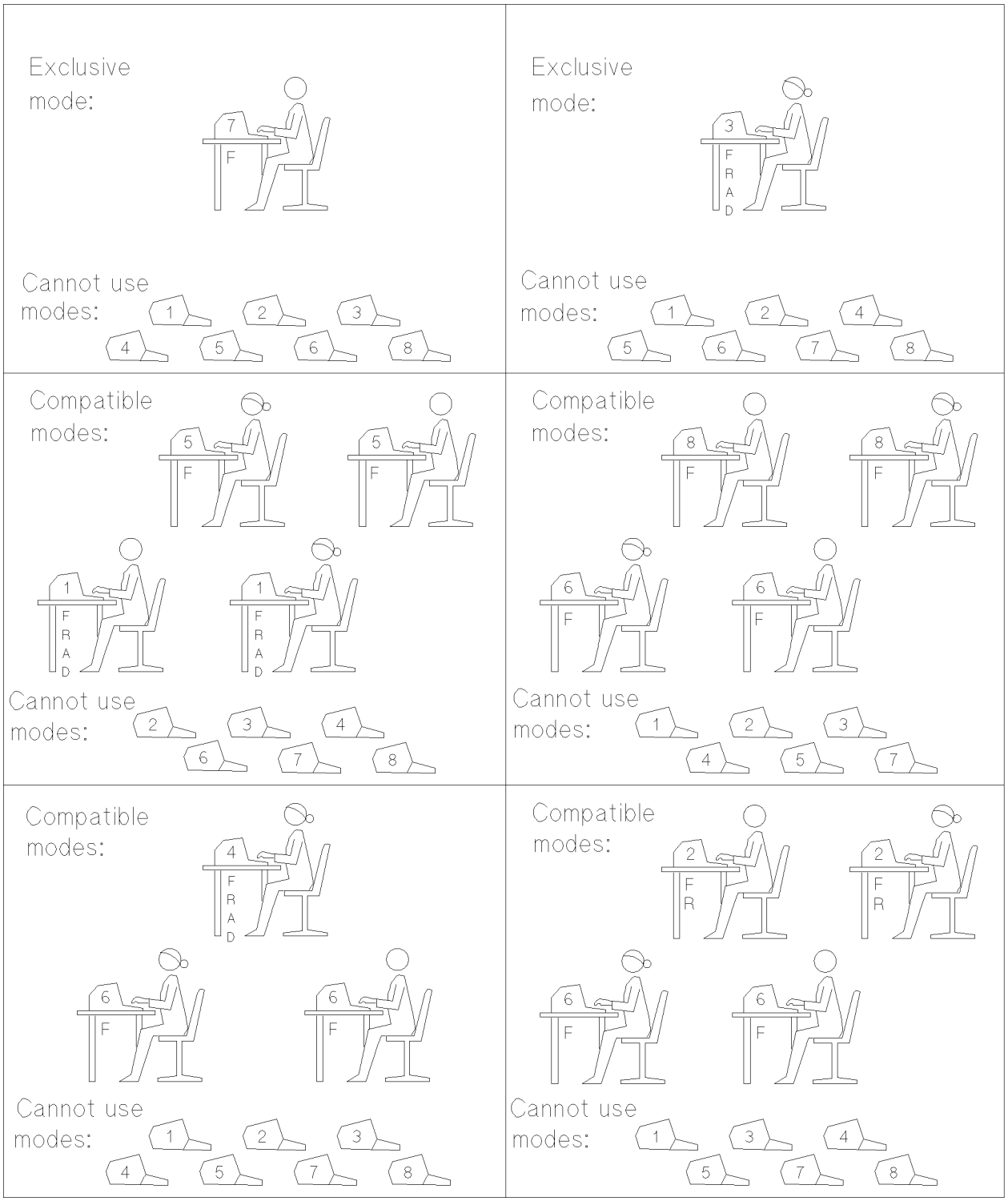


Figure 1-3. IMAGE/QUERY Environments

Logging and Mode Selection

Opening the data base in modes 1 through 4 allows data base modifications to be logged to a log file and subsequently recovered if there is a system failure. If the data base administrator has enabled the data base for logging, certain requirements of the logging system must be met before the data base can be accessed in modes 1 through 4.

These requirements are the responsibility of the data base administrator and console operator, and are discussed in the *IMAGE Reference Manual*. If any of these requirements are not satisfied, an IMAGE error message is returned.

PASSWORDS

Before you can access a data base, you must specify your password. The password will not be echoed back to the screen after entering in session mode nor will it appear if QUERY is in job mode. The password you specify determines the access you have to the data items and data sets in the data base. If you are logged on as the data base creator and use the semicolon (;) password, you will have read and write access to all data items and data sets in the data base. This is true even if no passwords are defined for the data base.

In some cases, it is possible to gain access to some data without a password if the data base is designed to allow such access. Passwords are defined by the data base designer or administrator and control read and write access to information in a data base. You must ask the data base administrator for a password and enter it as described in Section 2.

Note: In this manual, entering a password is illustrated with the password underlined. However, during an actual session the password will not appear on the terminal screen.

QUERY CHARACTER SET

The following ASCII characters are defined as the QUERY character set: A through Z, a through z, 0 through 9, and +-*/?'#&@%

All other characters are referred to as special characters. When you encounter the term special characters in the following sections, it refers to all characters which are not part of this set. Blanks are special characters. The complete ASCII character set is shown in Appendix G.

Lowercase characters are always upshifted (changed to upper case) unless they are part of a literal or values for a data item of type X.

MPE FILES

For those readers who are not familiar with the MPE Operating System, here is a brief introduction to some MPE concepts you may need to understand while using QUERY.

MPE treats an input/output device as a file with a standard name. For example, $STDLIST is the name of the standard output file for both session and job modes. A file is equated to a device class which may be a line printer, terminal, disc, or other peripheral device. $STDLIST is normally equated to a terminal in session mode and to a line printer in job mode. The system manager defines the various device classes and sets up the correspondence between device classes and standard files.

QUERY uses a file named QSLIST to allow you to alter the normal output device to any other output device. You must use an MPE :FILE command to equate QSLIST to the device class you select. (The default device class is LP.) You then use the QUERY OUTPUT= command to change the output file from $STDLIST to QSLIST. The method for doing this is explained in detail with the OUTPUT= command in Section 3. The system manager can tell you what various device types are on your system.

MPE ASCII files contain ASCII characters and may be in disc or magnetic tape format. You can use EDIT/3000, TDP/3000, or any editor that accesses ASCII files, to create these files.

2

USING QUERY/V

QUERY can be run in either session mode (interactive) or job mode (batch). In session mode, QUERY issues command prompts, error messages, and other messages of general interest to you. The manner in which QUERY proceeds depends upon your responses to QUERY messages and prompts. In job mode, QUERY reads commands and other input from a job file and prints messages on the $STDLIST device.

Before accessing data in either session or job mode, you must define the QUERY environment. After you define the environment, you use QUERY to enter, replace, delete, retrieve, and report data in the data base(s).

You can save a frequently used command as a named procedure or a sequence of commands as an XEQ file. Procedures and XEQ files allow you to execute commands repeatedly without re-entering the commands. These features can be used in either session or job mode.

Note: In this manual, user input, including passwords, is underlined when necessary for clarity. In an actual session, the password will not be echoed back to the screen. Also in this manual, examples are shown in uppercase letters. When using QUERY, you may use either upper or lowercase letters. Passwords must be entered exactly as defined by the data base administrator.

USING QUERY IN SESSION MODE

When entering information in session mode, you can delete one or more previously typed characters by entering a CONTROL H character for each character you want to delete. You can delete the current line with CONTROL X. Control characters are entered by pressing the control key while pressing the appropriate letter key.

Running QUERY

The following illustrates the procedure to initiate a QUERY session.

```
RETURN
:HELLO MANAGER.DATAMGT,ACCOUNTS
HP3000 / MPE V  G.A2.A0 (BASE G.A2.A0).  MON, MAR 23, 1987, 11:41 AM

:RUN QUERY.PUB.SYS
HP32216C.00.08 QUERY/V  MON, MAR 23, 1987, 11:42 AM
COPYRIGHT HEWLETT-PACKARD CO. 1976

>
```

Log on to the computer using the MPE :HELLO command. MPE prints information about your session and another colon. Initiate QUERY using the MPE :RUN command. QUERY prints the subsystem banner and prompts for a command with >.

Accessing Data

Before you can access data, you must open a data base by specifying a data base name, password, and access mode. You can do this at the beginning of your QUERY session by using the DATA-BASE= or DEFINE command, both of which will prompt you with >> to supply information. For example:

```
>DATA-BASE=ORDERS
PASSWORD = >>CLERK
MODE = >>5
>
```

You may open additional data bases with the MULTIDB command after you specify the primary data base with the DATA-BASE= or DEFINE command. The data base opened with the DATA-BASE= or DEFINE command is the primary data base. The primary data base is used by QUERY when executing a command unless a data base opened with the MULTIDB command is specified. You can change the primary data base at any time with the DATA-BASE= or DEFINE command.

After opening the data base(s), QUERY can be used to add, delete, replace, locate, and/or report data in the data base(s). During QUERY execution, you may need to temporarily return to the operating system to use the MPE :FILE command to direct output to a specific device such as a line printer.

Using the Break Key

You use the break key to temporarily return to the operating system. After you have finished entering MPE commands, you return to QUERY by using the MPE :RESUME command. MPE prints READ PENDING to inform you that QUERY is waiting for a command. In this case, QUERY does not print the > prompt until you press (RETURN). For example:

```
>(BREAK)
:FILE QSLIST;DEV=TAPE
:RESUME
READ pending
(RETURN)
>
```

Using Control Y

Some QUERY commands which must search multiple data sets or data bases may take some time. If you do not want to wait until a command execution is completed, you can terminate the command with (CONTROL) Y. QUERY will print < CONTROL Y > and prompt for a new command. If (CONTROL) Y is used during the execution of a retrieval command (FIND, MULTIFIND, or SUBSET), QUERY will print the number of entries found and ask if you wish to continue searching for entries. Respond "NO" to terminate the command. For example:

```
>FIND LAST-NAME IS MORGAN
```

```
USING SERIAL READ
(CONTROL) Y
< CONTROL Y >
2  ENTRIES HAVE QUALIFIED,
DO YOU WANT TO CONTINUE SEARCHING? NO
>
```

Exiting QUERY

When you are finished using QUERY, you use the QUERY EXIT command to end QUERY execution. To terminate your MPE session, use the MPE :BYE command. MPE responds by printing the number of seconds of CPU time used and the number of minutes you were connected to the system.

```
>EXIT

END OF PROGRAM
:BYE

CPU=6.  CONNECT=6.  TUE, JAN 7, 1986, 2:05 PM
```

USING QUERY IN JOB MODE

In job mode, QUERY reads commands and other input from the job file. Output from the job and QUERY messages are printed on the device designated as $STDLIST, which is usually a line printer.

Only the first 72 characters of each line (record) are read by QUERY. Any characters on the line beyond 72 characters are ignored. If a QUERY command is longer than 72 characters, you can continue it on the next line by using an ampersand (&) as the last non-blank character on the line to be continued.

As in session mode, you must open a primary data base before you can access data. You must do this at the beginning of your job file. QUERY commands which access the data base must follow the identification of the primary data base. If QUERY is unable to open the data base for any reason, the job will terminate.

You must anticipate the order of the prompts issued by some QUERY commands, such as DATA-BASE=, and supply the information on separate lines in the job file. If a command is out of sequence or incorrectly entered, QUERY issues an error message and, in some cases, terminates the job.

Some commands operate differently in job mode. These differences in command operation are discussed under the specific command in Section 3.

The following example shows a job file used to operate QUERY in job mode.

```
:EDITOR
HP32201A.7.15 EDIT/3000 MON, MAR 23, 1987 11:49 AM
(C) HEWLETT-PACKARD CO. 1985

/TEXT EXAMPLE
/LIST ALL
     1     :JOB EXAMPLE,MANAGER.DATAMGT,ACCOUNTS
     2     :RUN QUERY.PUB.SYS
     3     DATA-BASE=ORDERS
     4     CLERK
     5     5
     6     DATA-SETS=CUSTOMER
     7     FIND LAST-NAME IS MARTENSEN
     8     REPORT ALL
     9     EXIT
    10     :EOJ
/EXIT
:STREAM EXAMPLE
 #J539
:
```

Figure 2-1. Job Mode Operation

Discussion

In a job file, the QUERY prompts (> and >>) must be excluded. Only MPE commands require prompts (:).

The first line of the job file must contain an MPE :JOB command. The second line contains an MPE :RUN command to initiate QUERY execution.

On line 3, the data base name, ORDERS, is specified using the DATA-BASE= command. The DEFINE command could also have been used to open the primary data base. In session mode the DATA-BASE= command prompts for the password and access mode. In job mode, this information must be supplied on separate lines in the correct order and without the prompts. In this example, the password is CLERK and the access mode is 5.

The data set name, CUSTOMER, is specified for the data set list using the DATA-SETS= command. The FIND command with retrieval specifications is used on line 7. The REPORT ALL command is used to display the entries which qualified.

QUERY execution is terminated with the EXIT command on line 9. The job is terminated with an MPE :EOJ command (end of job) on line 10. The MPE :STREAM command is used to execute a job file. MPE assigns and displays a job number.

Refer to the *MPE Commands Reference Manual* for more information about running programs in job mode.

QUERY COMMANDS

QUERY commands are divided into seven categories according to their function. The categories can be summarized as follows:

CATEGORY	FUNCTION
Environment	Defines data base(s) and conditions for QUERY execution.
Retrieval	Locates information to be used by other commands.
Reporting	Prints specified information.
Updating	Modifies the data base(s).
Procedure	Operates on procedures in the current Proc-file.
Operation	Operates on XEQ files.
Utility	See below.

Utility commands are used for a variety of tasks. The FORM, SHOW, and VERSION commands show the current state of QUERY. The REDO and LISTREDO commands allow you to list, edit, and re-execute a previous command. The ASSIGN, RELEASE, and SETLOCKS commands operate on the data set or data base locking option. The TRANSBEGIN, TRANSEND, and TRANSMEMO commands operate on the transaction logging facility. SAVE and UNDO operate on the internal select file used by QUERY during execution. The HELP command is an interactive facility used to determine the syntax and function of QUERY commands. EXIT is used to terminate QUERY execution.

Table 2-1 lists all the QUERY commands and their functions by category. The commands are discussed in detail in Section 3.

Table 2-1. Command Categories and Functions

CATEGORY	COMMAND	FUNCTION
Environment	CLOSE	Closes a data base opened with the MULTIDB command.
	DATA-BASE=	Specifies the primary data base, password, and access mode.
	DATA-SETS=	Specifies the data set list for the primary data base.
	DBLIST=	Informs QUERY which data bases to access for multiple data set retrieval and reporting.
	DEFINE	Specifies or shows the current QUERY environment for the primary data base.
	JOIN	Defines the compound data set used for multiple data set retrieval.
	JOIN PROCEDURE	Executes a JOIN procedure stored in the current Proc-file.
	LANGUAGE=	Shows the user language.
	MODE=	Specifies mode of access to the primary data base.
	MULTIDB	Used to open additional data bases and define or show their environment specifications.
	OUTPUT=	Specifies the output device.
	PASSWORD=	Specifies the password to access the primary data base.
	PROC-ENTITY =	Specifies the name of the current Proc-file.
Retrieval	FIND	Locates data entries in the data base.
	FIND ALL	Locates all entries in the data set regardless of the data item value specified.
	FIND CHAIN	Locates data entries from a detail data set and one or more of its corresponding master data sets.
	FIND PROCEDURE	Executes a FIND procedure stored in the current Proc-file.
	MULTIFIND	Retrieves compound data entries from a compound data set specified by the most recent JOIN command.
	MULTIFIND ALL	Retrieves all compound entries from a compound data set specified by the most recent JOIN command.
	MULTIFIND PROCEDURE	Executes a MULTIFIND procedure stored in the current Proc-file.
	SUBSET	Retrieves data entries from a FIND or MULTIFIND.
	SUBSET PROCEDURE	Executes a SUBSET procedure stored in the current Proc-file.

Table 2-1. Command Categories and Functions (continued)

CATEGORY	COMMAND	FUNCTION
Updating	UPDATE ADD	Adds data entries to the data base.
	UPDATE DELETE	Deletes data entries from the data base.
	UPDATE REPLACE	Modifies the values of data items.
	UPDATE PROCEDURE	Executes an UPDATE ADD, UPDATE DELETE or UPDATE REPLACE procedure stored in the current Proc-file.
Reporting	LIST	Prints data entries with automatic formatting.
	REPORT	Reports the data entries located in the previous retrieval.
	REPORT ALL	Prints data item values of entries located by the last retrieval command without formatting.
	REPORT PROCEDURE	Executes a REPORT procedure stored in the current Proc-file.
Operation	COMMENT	Allows comments to be added to an XEQ file.
	XEQ	Executes QUERY commands from an XEQ file.
Procedure	ALTER	Modifies a procedure stored in the current Proc-file.
	CREATE	Stores a command as a procedure in the current Proc-file.
	CREATE SPACE	Reports the number of unused records in the current Proc-file.
	DESTROY	Deletes a procedure from the current Proc-file.
	DISPLAY	Lists a procedure stored in the current Proc-file.
	DISPLAY LIST	Lists the names of the procedures in the current Proc-file.
	RENAME	Changes the name of a procedure in the current Proc-file.
Utility	ASSIGN	Allows a user to enable or disable the locking option.
	EXIT	Terminates QUERY execution.
	FORM	Lists information about the data bases currently being accessed.
	HELP	Lists information about the function, format, and parameters of QUERY commands.
	LISTREDO	Displays up to the last 20 commands issued.
	REDO	Displays the specified command for editing and executes the edited command.
	RELEASE	Removes all locks set by the SETLOCKS command.
	SAVE	Saves the retrieved data entries in a self-describing file.
	SETLOCKS	Prevents automatic unlocking of a data set.

Table 2-1. Command Categories and Functions (continued)

CATEGORY	COMMAND	FUNCTION
	SHOW	Displays the current data base list, JOIN, LOCKOPTION, and user language.
	TRANSBEGIN	Marks the beginning of a logical logging transaction.
	TRANSEND	Marks the end of a logical logging transaction.
	TRANSMEMO	Allows a message to be written to the log file.
	UNDO	Restores the entries retrieved from the previous FIND, MULTIFIND or SUBSET.
	VERSION	Displays the current version of QUERY and lists the IMAGE procedures and program files.

IDENTIFYING THE QUERY ENVIRONMENT

The three commands used to open data bases are DATA-BASE=, DEFINE, and MULTIDB. All of these commands prompt for various environment specifications with two greater than symbols (>>).

The DATA-BASE= or the DEFINE command is used to open the primary data base. You can use either command, but only one data base can be the primary data base at any time. You use the DATA-BASE= command to open the primary data base if you only want to open a data base. The DATA-BASE= command prompts for the password and access mode. You must specify a password (if the data base has security) and an access mode.

```
>DATA-BASE=ORDERS
PASSWORD = >>CLERK
MODE = >>5
>
```

You use the DEFINE command to open the primary data base if you want to define additional environment specifications. The DEFINE command prompts you for the password and access mode, as well as the data set list, procedure file, and output device. You may press RETURN in response to these additional prompts or you may supply information. If you do not supply information, you can define these specifications later by using the DATA-SETS=, PROC-ENTITY =, and OUTPUT= commands. The default output device is your terminal. You may define a different output device by responding to the OUTPUT=.

```
>DEFINE
DATA-BASE = >>ORDERS
PASSWORD = >>CLERK
MODE = >>1
DATA-SETS = >>CUSTOMER
PROC-ENTITY = >>MANPROC
OUTPUT = TERM
OUTPUT = >>RETURN
>
```

After you have opened a primary data base using the DATA-BASE= or the DEFINE command, you can open additional data bases using the MULTIDB command. The MULTIDB command prompts for the password, access mode, and data set list. You can press RETURN at the DATA-SET prompt. You can open multiple data bases with one MULTIDB command. To terminate the command, press RETURN at the DATA-BASE= prompt.

```
>MULTIDB
DATA-BASE = >>CITY
PASSWORD = >>OPEN
MODE = >>3
DATA-SETS = >>DISTRICT
DATA-BASE = >>STATE
PASSWORD = >>USE
MODE = >>3
DATA-SETS = >>COUNTY
DATA-BASE = >>RETURN
>
```

Both the DEFINE and MULTIFIND commands can also be used to display the current environment specifications. Each of the environment prompts can also be used as a command to change the specifications. For information on the function of each prompt, refer to the corresponding command.

LOCATING DATA

The retrieval commands, FIND, MULTIFIND, and SUBSET, locate entries in the data base(s) according to data item values in the entries. You can use retrieval commands to:

- Locate entries in a single data set.
- Locate entries in multiple data sets.
- Locate all entries in one data set or multiple data sets regardless of the value of the data item specified.
- Locate entries in multiple data bases.

Entries must be located before they can be reported, replaced, or deleted. The located entries remain available for use until QUERY execution is terminated, the entries are deleted, or until another FIND, MULTIFIND or JOIN command is entered. This means that located entries can be listed using one report format and then again using another format without using the retrieval command again.

A retrieval command consists of one or more relations, each containing a data item name, relational operator, and one or more values separated by commas. For example:

```
>FIND LAST-NAME IS MARTENSEN
USING SERIAL READ
1  ENTRIES QUALIFIED
>
```

In this example, L-NAME is the item name, IS is the relational operator, and MARTENSEN is the value. QUERY prints the number of qualifying entries.

When a retrieval command is executed, QUERY searches the appropriate data set(s) for the data entries which satisfy the relation(s). QUERY then stores the record addresses of these entries in a select file. When a REPORT, REPORT ALL, UPDATE DELETE or UPDATE REPLACE command is executed, the select file is used to locate the data entries.

A retrieval command can be stored as a procedure in a Proc-file for repeated use without re-entering the command. Retrieval procedures can be created which prompt you for the desired search values when the procedure is executed. This allows you to search for different values of the same data item each time the procedure is executed.

MODIFYING DATA

The updating commands, UPDATE ADD, UPDATE DELETE, and UPDATE REPLACE, change the data in the data base. UPDATE DELETE and UPDATE REPLACE operate on the entries selected by the last FIND, FIND ALL, or SUBSET command following a FIND or FIND ALL command. UPDATE ADD does not require a previous retrieval command.

You can modify your data base in three ways:

- Add data entries to a data set.
- Delete data entries from a data set.
- Change the value of data items not defined as search or sort items of the data set.

You must have specified an access mode of 1, 2, 3, or 4 to use the updating commands. If you opened the data base with another mode, you can use the MODE=, DEFINE, DATA-BASE=, or MULTIDB command to specify another mode. If you opened the data base with mode 2, you can only use UPDATE REPLACE. If you specify mode 1, 3, or 4, can use any of the updating commands. However, if the data base administrator has disallowed data base modification, no one can modify the data base, regardless of the mode specified.

```
>DEFINE
DATA-BASE = >>ORDERS
PASSWORD = >>CLERK
MODE = >>2
DATA-SETS = >>CUSTOMER
PROC-ENTITY = >>RETURN
OUTPUT = TERM
OUTPUT = >>RETURN
>FIND LAST-NAME IS MARTENSEN
USING SERIAL READ
1  ENTRIES QUALIFIED
>UPDATE REPLACE,STREET-ADDRESS="2451 CHASEN ROAD";END
>
```

An updating command can be stored as a procedure in a Proc-file for repeated use without re-entering the command. Updating procedures can be created which prompt you for the values to be added, deleted, or replaced when the procedure is executed. This allows you to modify the data with different values for the same data item(s) each time the procedure is executed.

REPORTING DATA

The reporting commands, LIST and REPORT, display selected information from the data base(s). QUERY provides several reporting techniques which enable you to examine the data in the data base without writing a program. You can report information in three ways:

- Use one command to retrieve data entries for a single report with automatic formatting.
- Use two commands, one to retrieve data entries which can be used repeatedly and another to produce a report with automatic formatting.
- Use two commands, one to retrieve data entries which can be used repeatedly and another to produce a report with your formatting specifications.

The LIST command combines the functions of locating and reporting. You can selectively locate entries and print them in a report with automatic formatting and headings. Data in the formatted report is printed in columns.

REPORT ALL is a form of the REPORT command. REPORT ALL is similar to the LIST command in that it automatically formats your report. However, REPORT ALL reports data entries selected by the last retrieval command. The formatted report prints one data item name and value per line for all data items in each retrieved entry.

The REPORT command also reports data entries selected by the last retrieval command. However, the REPORT command allows you to design your own report format. Your report can include:

- From 1 to 9 lines of heading information such as a title, column headings, page numbers, date and time of day. This information can be repeated at the top of each report page. You can specify blank lines to be interspersed in the heading as long as the heading does not exceed one page.
- Sorted data entries with group and total information printed with entries belonging to the same group. You can accumulate totals, compute averages, and count entries automatically, or you can use the thirty QUERY registers to do computations on numeric data item values and report the results.
- Edited data item values with inserted dollar signs, minus signs, decimal points, and other ASCII characters.
- Statements which change the output device to the QSLIST device, define the number of lines per page, request a pause after each page (in session mode), and suppress the margins which usually appear at the top and bottom of a page. (For example, information on a line printer can be printed across the perforation.)

The REPORT command can be stored as a procedure in a Proc-file for repeated use without re-entering the command. Reporting procedures can be created which prompt you for desired information when the procedure is executed. This allows you to use different specifications for the report each time the procedure is executed.

You can also write your own user-defined procedures in a programming language which enable your report to perform specialized functions not provided by QUERY. This feature is intended for use by programmers. Refer to Appendix F for further discussion.

USING PROCEDURES IN A PROC-FILE

A single QUERY command can be stored as a named procedure for repeated use without re-entering the command. The commands which can be used in a procedure are listed in Table 2-2.

You execute a procedure by referencing it with a form of the command used in the procedure. Refer to Table 2-2 for the commands that are used to execute procedures. For example, if a procedure named FINDNAME contains the FIND command, you would execute the procedure by typing:

```
>FIND FINDNAME
```

You store one or more procedures in an MPE file called a Proc-file. Only one Proc-file is "active" at any time. The active Proc-file is called the current Proc-file. When you execute a procedure, QUERY searches the current Proc-file for the specified procedure. Therefore, before executing a procedure, you must specify the current Proc-file with the PROC-ENTITY = command or in response to the prompt issued by the DEFINE command. To access a Proc-file, you must have read and lock access to the group and account in which the Proc-file resides.

Table 2-2. Procedure Commands

USED IN A PROCEDURE	EXECUTES A PROCEDURE
FIND FIND ALL FIND CHAIN	FIND *procedure*
JOIN	JOIN *procedure*
MULTIFIND MULTIFIND ALL	MULTIFIND *procedure*
REPORT REPORT ALL	REPORT *procedure*
SUBSET	SUBSET *procedure*
UPDATE ADD * UPDATE DELETE * UPDATE REPLACE *	UPDATE *procedure*

* In a procedure, the updating commands must include the UPDATE keyword or its abbreviation.

A procedure can be created within QUERY or copied from an MPE ASCII file. The QUERY commands that operate on procedures are discussed in Table 2-3.

Table 2-3. Commands Used to Define Procedures

COMMANDS	FUNCTION
CREATE	Creates a procedure or copies it from an MPE ASCII file and stores it in the current Proc-file. The CREATE SPACE option shows the number of unused records in the current Proc-file.
DISPLAY	Lists the names of the procedures in the current Proc-file or lists individual procedures with line numbers for use in editing.
ALTER	Edits a procedure stored in the current Proc-file. This command is used to insert, replace, and delete lines.
DESTROY	Deletes a procedure from the current Proc-file.
RENAME	Changes the name of a procedure in the current Proc-file.

USING QUERY COMMANDS FROM AN XEQ FILE

A sequence of QUERY commands can be named and stored in an MPE ASCII file, referred to as an XEQ file, for repeated use without re-entering the commands. The XEQ command is used to execute an XEQ file. For example, you execute an XEQ file named LISTNAME by typing:

```
>XEQ LISTNAME
```

An XEQ file must be created using EDITOR, TDP, or any editor which operates on ASCII files. You must exit QUERY to create an XEQ file.

The following is a list of differences between an XEQ file and a procedure:

- Only the commands listed in Table 2-2 can be used in a procedure, but any QUERY command can be used in an XEQ file.
- Only one command can be stored in a procedure, while an XEQ file can contain a sequence of commands.
- A procedure can be modified from within QUERY, while an XEQ file can only be modified using an editor outside of the QUERY subsystem.
- An XEQ file is a unique file which can be listed using the MPE :LISTF command. However, a procedure is contained in a Proc-file and can only be listed within QUERY using the DISPLAY command.
- The current Proc-file must be specified before a procedure can be executed, but an XEQ file can be executed at any time.

Refer to the XEQ command in Section 3 for more information on using XEQ files.

3

QUERY/V COMMANDS

The following characteristics apply to all commands.

- Command names can be spelled out completely or abbreviated. The minimum abbreviation is specified in the syntax under each command. In most cases, any subset of the characters between the full command name and the abbreviation may be used. For example, the syntax of the FORM command specifies FO[RM]. In this case, any of the following may be used: FORM, FOR, or FO. Refer to "Conventions Used in This Manual" for an explanation of the symbols used in the syntax in this manual.
- Commands consist of English keywords and parameters (both required and optional).
- QUERY processes only the first 72 characters of a line (record). Any remaining characters can be used for comments or sequencing information.
- If the command you want to enter is longer than 72 characters, you can continue it on the next line by using an ampersand (&) as the last character on the current line. QUERY combines all lines connected with the & continuation character. Any blanks preceding the & are saved. Therefore, if you break a command name or other parameter with an &, the & should be adjacent to the last significant character with no intervening blanks. If a blank is necessary, it can be included at the beginning of the next line.

Refer to Table 2-1 for a list of QUERY commands and their functions.

Note During an actual session, passwords are not echoed back to the screen. The examples in this manual are shown in uppercase letters. When using QUERY, you can use either lower or uppercase letters except for passwords which must be entered exactly as defined by the data base administrator.

ALTER

Inserts, replaces, and deletes lines of a procedure stored in the current Proc-file.

Syntax

A[LTER] *procedure name*

For example:

```
>ALTER REP22
```

Where *procedure name* = REP22

Parameter

procedure name — is the name of a procedure stored in the current Proc-file.

Discussion

When you enter the ALTER command, QUERY prompts you for insert, replace, and delete statements by printing >>. Each statement operates on a line or range of lines in the procedure. In the statement descriptions that follow, m is the first line number in the range, and n is the last line number. n must always be greater than or equal to m, and m must be greater than or equal to 1. Neither m nor n can exceed the total number of lines in the procedure.

Once you enter the ALTER command, you must do your editing sequentially from lower numbered lines to higher numbered lines. You cannot go back to a line which precedes the ones you are currently editing without exiting the ALTER command and entering another ALTER command. This technique is illustrated in the examples which follow.

If an error occurs after a statement is entered, an error message is printed and QUERY prompts you for another ALTER statement.

You can abort the command at any time by entering CONTROL Y. If you do this, the procedure will remain unchanged. Always terminate the command with /E if you want to save the results of your changes.

You can use one of four statements in response to an ALTER prompt: Insert, Replace, Delete, and End.

Lines which are added or changed are not checked for correct syntax until the procedure is executed. Similarly, if line deletion causes a command to have incorrect syntax, it not will be detected until the procedure is executed.

ALTER - INSERT STATEMENT

Inserts lines into the procedure. QUERY prompts for each statement and line with >>.

Syntax

```
>>/I, m  >>line  >>line  .   .
```

Parameters

m	is the number of the procedure line after which you want to insert statements.
line	is the new line you want to insert.

Discussion

The lines which follow the insert statement are inserted into the procedure following line number *m*. You cannot insert lines in front of the first procedure line. To indicate you do not want to insert mores lines, enter a slash followed by another ALTER statement.

Example

```
>ALTER FIND1
>>/I,1
>>STOCK# IS "" AND
>>LASTSHIPDATE IS ""
>>/E
```

The example above changes the procedure FIND1 from:

```
text
      001  FIND
      002  END

To:

      001  FIND
      002  STOCK# IS "" AND
      003  LASTSHIPDATE IS ""
      004  END
```

ALTER - REPLACE STATEMENT

Deletes the specified line or range of lines and replaces them with the lines following the replace statement. QUERY prompts for each statement and line with >>.

Syntax

>>/R,*m*[,*n*] *(current lines m through n are listed)* >>*replacement line* >>*replacement line* . .

Parameters

m	is the number of the first procedure line you want to replace.
n	is the number of the last procedure line you want to replace. If *n* is not specified, only line *m* is replaced.
replacement line	is the new line which will replace the existing line.

Example

```
>>ALTER UPDATE1
>>/R,2,3
HOURS="";
DATE="851022";
>>HOURS="14";
>>DATE="851101";
>>STATUS="OK";
```

The example above changes the procedure UPDATE1 from:

```
text
      001  UPDATE REPLACE,
      002  HOURS="";
      003  DATE="851022";
      004  END

To:

      001  UPDATE REPLACE,
      002  HOURS="14";
      003  DATE="851101";
      004  STATUS="OK";
      005  END
```

ALTER - DELETE STATEMENT

Deletes the specified line or range of lines. QUERY prompts for each statement with >>.

Syntax

>>/D,*m* [,*n*] *(deleted lines are listed)*

Parameters

m	is the first line you want to delete.
n	is the last line you want to delete. If *n* is not specified, only line *m* is deleted. If line *n* is specified, the range of lines from *m* to *n* is deleted.

Discussion

If you try to delete all the lines in the procedure, an error message is printed. To delete the entire procedure, use the DESTROY command.

Example

```
text
>ALTER REPORT4
>>/D,4,5
S2,ORDERDATE
S,LASTSHIPDATE
>>/E
```

The example above changes the procedure named REPORT4 from:

```
      001  REPORT
      002  H1,"MONTHLY SHIPMENTS",25,SPACE A2
      003  S1,STOCK#
      004  S2,ORDERDATE
      005  S,LASTSHIPDATE
      006  G1,STOCK#,15
      007  END
```

To:

```
      001  REPORT
      002  H1,"MONTHLY SHIPMENTS",25,SPACE A2
      003  S1,STOCK#
      004  G1,STOCK#,15
      005  END
```

ALTER - END STATEMENT

Terminates the ALTER command and saves the procedure.

Syntax

```
>>/E
```

Discussion

QUERY continues to prompt you for insert, delete, or replace statements until you enter /E. If the ALTER command is terminated using CONTROL Y, the entire command is ignored and the procedure remains in its original state. To save your changes, you must terminate the ALTER command with /E.

Example

```
text
>DISPLAY REPORT2

PROCEDURE:  REPORT2

001  REPORT
002  H1,"AS OF:",6
003  H1,DATE,15
004  H2,"BOBO'S MERCANTILE",45
005  H1,"PAGE",69
006  H1,PAGENO,71
007  D1,STOCK#,36
008  D1,LASTSHIPDATE,48,E2
009  E2,"XX/XX/XX"
010  END
```

The following is an example of using the ALTER command with all four statements. The DISPLAY command can be used to examine the procedure and the results of the ALTER command.

```
>ALTER REPORT2
>>/R,2,3
H1,"AS OF:",6
H1,DATE,15
>>H1,"TO DATE:",6
>>H1,DATE,20
>>/D,5,6
H1,"PAGE",69
H1,PAGENO,71
>>I/,9
>>S,LASTSHIPDATE
>>/E
>

>DISPLAY REPORT2

PROCEDURE: REPORT2
```

```
001   REPORT
002   H1,"TO DATE:",6
003   H1,DATE,20
004   H2,"BOBO'S MERCANTILE",45
005   D1,STOCK#,36
006   D1,LASTSHIPDATE,48,E2
007   E2,"XX/XX/XX"
008   S,LASTSHIPDATE
009   END
```

ASSIGN

Enables or disables the data base lock option.

Syntax

AS[SIGN][*data base name*:]LOCKOPTION = { ON / OFF }

For example:

```
>ASSIGN LOCKOPTION = OFF
```

Parameters

data base name	is the name of a data base opened with either DEFINE, DATA-BASE=, or MULTIDB. You must specify the data base name if more than one data base is open.
ON	enables the lock option. The default ON.
OFF	disables the lock option.

Discussion

When LOCKOPTION is set to ON (the default), QUERY automatically locks data bases opened by QUERY in mode 1 or 5 while a retrieval or reporting command is executing. This ensures that the resulting report does not contain errors due to dynamic changes made by other users concurrently accessing the same data.

In some cases, automatic locking may be overprotective if no other users can concurrently modify data that affects your retrieval or reporting command. For example, protection may be provided through data base passwords or through exclusive scheduling of tasks. In such cases, automatic locking by QUERY can needlessly increase response time for other users. By setting the LOCKOPTION to OFF, you prevent QUERY from locking data bases opened in mode 1 or 5. However, when updating a data base opened in mode 1 or 5, QUERY locks regardless of the state of the LOCKOPTION.

The LOCKOPTION can be set to OFF or ON at any time, and remains in effect until it is reset by a subsequent ASSIGN command. To determine the current state of the LOCKOPTION, use the SHOW command.

CLOSE

Closes a data base opened with the MULTIDB command.

Syntax

CL[OSE] *data base name*

For example:

```
>CLOSE ORDERS
```

Parameter

data base name	is the name of a data base opened with the MULTIDB command.

Discussion

The CLOSE command allows you to close a data base opened with the MULTIDB command, without exiting from QUERY. The data base that was opened with DEFINE or DATA-BASE= commands is unaffected and remains open until the end of your job or session.

You may want to close a data base opened with a MULTIDB command for the following reasons:

1. You have reached the limit of 10 open data bases that QUERY allows.
2. You want to re-open your data base with the DEFINE or DATA-BASE= command so that it will be the primary data base.
3. You have exclusive access and/or have finished using the data base and want to allow access to other users.

A data base closed with the CLOSE command will not be listed with subsequent MULTIDB commands. Data items in a closed data base can no longer be accessed. Any data bases not closed with the CLOSE command are automatically closed by QUERY at the end of the job or session.

COMMENT

Allows you to add comments to an XEQ file.

Syntax

CO[MMENT] *text*

Parameter

text is a string of up to 698 characters. A comment can be continued with an ampersand.

Discussion

The COMMENT command cannot be embedded with the DEFINE, DATA-BASE=, REPORT, ALTER, or UPDATE ADD commands since it will be interpreted as input.

Example

```
COMMENT  ------------------------------------------------&
         This retrieves all the accounts in Salt Lake City&
         and reports the account number and shipping date.
COMMENT  ------------------------------------------------

FIND ACCOUNT.CITY="05"
REPORT
H1,"SALT LAKE CITY ACCOUNTS:",50
H2," ",50,SPACE A2
H1,DATE,70
S,ACCTNUM
D1,ACCTNUM,20
D1,SHIPDATE,40
END
```

CREATE

Stores a command as a named procedure in the current Proc-file.

Syntax

C[REATE] *procedure name*, { *file name* | *command* END }

For example:

```
>CREATE FIND6,FINDFILE

>CREATE FINDNAME,FIND LAST-NAME IS "" END
```

Parameters

procedure name	is a name composed of 1 to 8 alphanumeric characters chosen by you. The first character must be alphabetic. No special characters or spaces are allowed. The procedure name cannot be any of the following: ALL, D, DELETE, D*n*, E*n*, END, EZ, G*n*, H*n*, LIST, LP, NOPAGE, PAUSE, R*n*, S, S*n*, SPACE, TF, or T*n*, where *n* is an integer from 0 to 9. (QUERY only recognizes these as options or report statements, not as procedure names.)
file name	is the name of an MPE ASCII file containing one of the commands listed in Table 2-2. The command will be stored as a procedure in the current Proc-file. The *file name* cannot be FIND, REPORT, UPDATE, or any legal abbreviation of these commands. (QUERY cannot distinguish these names from the *command* parameter.) FINDX and other supersets are acceptable.
command	is one of the commands listed in Table 2-2. Refer to the command in this section for the syntax of the command.

Discussion

You can create a procedure in two ways:

- from input stored in an MPE ASCII file
- from input entered through the session or job input device

If the procedure you are creating does not fit in the available Proc-file space, an error message is printed. The incomplete procedure is stored, and you can list it with the DISPLAY command.

File Input

If you create the procedure from a file, QUERY reads the named file when the command is entered. QUERY then lists the file on the standard list device and stores it as a procedure in the current Proc-file. When the input is copied from the MPE ASCII file to the Proc-file, the procedure is not checked for correct syntax. The command syntax is not checked until the procedure is executed.

QUERY does not process the last eight characters of each record in an MPE ASCII file. If you create the file using EDIT/3000 and keep it with numbered lines, the last eight

characters are the line numbers. Even if you keep the file unnumbered, QUERY commands or parameters should not be entered in this part of a record.

Example

```
:EDITOR
HP32201A.7.17 EDIT/3000 MON, MAR 23,  1987  2:51 PM
(C) HEWLETT-PACKARD CO. 1983

/ADD
      1   FIND
      2   CUSTOMER.ACCOUNT IS ""
      3   END
      4   //
/KEEP FINDFILE
/EXIT

END OF SUBSYSTEM
:RUN QUERY.PUB.SYS
HP32216C.00.08  QUERY/3000  MON, MAR 23, 1987  2:51 PM
COPYRIGHT  HEWLETT-PACKARD CO.  1976

>DATA-BASE=ORDERS
PASSWORD = >>CLERK
MODE = >>1
>PROC-FILE =PROC2
>CREATE FINDACCT,FINDFILE
FIND
CUSTOMER.ACCOUNT IS ""
END
>
```

In the previous example, an MPE file is created with EDIT/3000. After running QUERY, the data base is opened with the DATA-BASE= command and the Proc-file is declared. When the CREATE command is issued, FINDACCT is created and listed on the screen.

Terminal Input

If you enter a procedure through a terminal (in session mode) or through a standard input device (in job mode), the command can be entered on one line or on as many lines as necessary without the use of the continuation character (&). Whenever you press RETURN, QUERY prompts for additional lines by printing >>. To terminate the command, enter a pair of slashes (//) or END as the last three characters in a line. All the characters you enter are stored in the current Proc-file without checking for correct syntax. The command is not checked for correct syntax until the procedure is executed.

QUERY allows a maximum input record of 250 characters. If you use the continuation character (&), QUERY considers all the lines connected with it as one input record.

Examples

```
>CREATE FINDNAME, FIND LAST-NAME IS "","","" END
>FIND FINDNAME
WHAT IS THE VALUE OF - LAST-NAME
>>MURTZ
WHAT IS THE VALUE OF - LAST-NAME
>>FRANZONI
WHAT IS THE VALUE OF - LAST-NAME
>>X
USING SERIAL READ
2  ENTRIES QUALIFIED
```

The previous example illustrates the result of creating a procedure with null values. When the procedure is executed, QUERY prompts for three LAST-NAME values. A value must be supplied for each null value. However, you can use a known invalid response if you do not want a third name. Refer to the FIND command for more information on using null values.

```
>CREATE CHECK,FIND CREDIT-RATING ILT 5 END
>FIND CHECK
USING SERIAL READ
2  ENTRIES QUALIFIED
>CREATE NAMES,
>>REPORT
>>D,LAST-NAME,20
>>//
>REPORT NAMES
>>PAUSE;
>>END

MCFALL
CELERY
```

The previous example shows two procedures. The second procedure, NAMES, operates on the results of the first procedure, CHECK. The procedure named CHECK is useful each time you want to find the names of customers with low credit ratings. The NAMES report procedure can also be used repeatedly to list names from the retrieved entries. As shown in the second procedure, it can be useful to leave the END keyword out of a REPORT procedure. When the procedure is executed, QUERY prompts for the missing END. You can then enter additional report statements, such as sort or output control statements, to vary the report output.

CREATE SPACE

Reports the number of unused records in the current Proc-file.

Syntax

```
C[REATE]SPACE
```

Discussion

You can use the CREATE SPACE command to find out how much space is left in the current Proc-file before you begin creating a new procedure. If you try to create a procedure that is larger than the space left in the current Proc-file, you will receive an error message and part of the procedure will be lost. If you know there is not enough space left, you can either create a new Proc-file, declare another Proc-file which has more space available, or delete unused procedures from the current Proc-file.

Each stored procedure starts on a record boundary and extends through as many records as necessary. Generally, one record is sufficient for most UPDATE and FIND commands. Long REPORT commands may take more than one record.

Example

```
>CREATE SPACE
RECORDS = 109
>DESTROY REPORT7
>CREATE SPACE
RECORDS = 100
```

In the previous example, nine records of space were gained when the procedure REPORT7 was deleted.

DATA-BASE=

Specifies the primary data base to be accessed by QUERY.

Syntax

[DATA-]B[ASE]= *data base name*

For example:

```
>DATA-BASE=ORDERS.PUB.SYS

>B=DBAFILE
```

Parameter

data base name — is the name of an IMAGE data base.

The data base may reside in any group or account on the local HP 3000 or a remote HP 3000 as long as you are allowed access to it through the MPE file security. To specify a data base that does not reside in your group and account, you must use a fully-qualified name in the form: *data base name.group.account.* For example, ORDERS.PUB.SYS is a data base named ORDERS in the PUB group of the SYS account.

Note: If you are accessing a remote data base using QUERY, refer to Appendix C.

Discussion

This command is used to specify the primary data base. You must use either this command or the DEFINE command to identify the primary data base before you can use QUERY commands which access a data base. You can specify a new primary data base at any time, and you can open additional data bases with the MULTIDB command.

When you enter the DATA-BASE= command, QUERY prompts you for a password and an access mode.

```
PASSWORD = >>
MODE = >>
```

When you enter this command, QUERY first closes the current primary data base before attempting to open the requested data base. If QUERY is unable to open the requested data base, you can:

- Specify a different data base with the DATA-BASE= or DEFINE command.
- Use the data base(s) opened with the MULTIDB command.
- Use commands which do not require an open data base (e.g., HELP).

Passwords and Access Modes

Passwords are created by the data base administrator or designer who will tell you which one you may use. The password determines which data sets and/or data items in the data base you are allowed to read and/or write. A password must be entered exactly as it was created, with including upper and lowercase characters.

If you are signed on as the creator and enter a semicolon in place of the data base password, you will be given read and write access to all data items and data sets in the specified data base. This is true even if there are no passwords specified for the data base.

If you enter an invalid password, you are assigned a user class of zero which will allow you read and/or write access to some or all of the data sets and data items in the specified data base. When this happens, the following message is returned:

```
PASSWORD DOES NOT MATCH ANY DEFINED FOR SPECIFIED DATA BASE;
USER CLASS ZERO (0) WAS ASSIGNED
```

If there are no data sets and/or data items which you can access, the following message will be returned:

```
BAD PASSWORD
```

If you are not sure that you will have access to the data items that you need, you can change the password with the DEFINE, DATA BASE=, or PASSWORD= command.

After you enter a password, QUERY prompts you for the access mode. You must enter a valid access mode represented by a number between 1 and 8. The description of modes in Section 1 will help you determine which mode to use.

Job Mode

If the DATA-BASE= command is entered in job mode, the password and the mode number must follow the command in the next two records, respectively.

Example

User 1

```
>DATA-BASE=ORDERS
PASSWORD = >>BUYER
MODE = >>1
```

User 2

```
>B=ORDERS
PASSWORD = >>CLERK
MODE = >>6
DATA BASE OPEN IN ANOTHER MODE

>B=ORDERS
PASSWORD = >>CLERK
MODE = >>5
>
```

This example shows two users for the ORDERS data base. User 2 must open the data base with mode 5 because mode 6 is incompatible with the access mode of user 1. User 1 has requested mode 1 (exclusive read and write access).

DATA-SETS=

Informs QUERY which data set to reference in the primary data base if a data item name which appears in more than one data set is used in a FIND, LIST, REPORT, or UPDATE command.

Syntax

[DATA-SET]S= [*data set list*]

For example:

```
>DATA-SETS=PAYROLL,ACCTREC

>DATA-SETS=
```

Parameter

data set list	is a list of data sets you want to access. The list can contain data set names and dummy data set names. Each name must be separated from the next name by a comma. If *data set list* is not included, the data set list is cleared.
A dummy data set	is a temporary data set used in the JOIN command. The dummy data set is cleared from the data set list when the next JOIN command is entered.

Discussion

The DATA-SETS= command defines a list of one or more data sets. If you use a FIND, MULTIFIND, SUBSET, REPORT, or LIST command with a data item name which is in more than one data set in the primary data base, QUERY will check the data set list to determine which data set to use.

In an IMAGE data base, different data items can have the same name if each appears in a different data set. As mentioned in Section 1, you can use a fully-qualified data item name (*data set name.data item name*) to tell QUERY which data item to use. However, if you are using fully-qualified names for multiple data items, the command could become quite lengthy. The DATA-SETS= command may save you some time.

Data Set Selection Rules

Session Mode

If you reference a data item appearing in more than one data set, QUERY resolves which data set to use according to the following rule:

- If one (and only one) of the data sets containing the data item appears in the data set list, QUERY automatically uses that data set. Otherwise QUERY prompts you to supply the desired data set with the message:

  data item name IS A MEMBER OF THESE SETS:
  data set name,data set name, ...
  WHICH SET DO YOU WISH TO USE?

  >>*data set name*

You must type the name of the data set you want to access. If the name you provide does not match the names listed, QUERY repeats the prompt. If you decide you do not want to access the listed data sets, you may abort the command by pressing RETURN instead of entering a data set name. You will be prompted for another command.

Job Mode

QUERY cannot prompt you for the desired data set in the event of ambiguity. The data set to be accessed is chosen according to the following rules:

- If exactly one data set containing the data item appears in the data set list, QUERY uses that data set.
- If more than one of the data sets containing the data item appears in the data set list, QUERY uses the last data set mentioned in the list.
- If no data set containing the data item is in the data set list, QUERY accesses the last data set in the data base that contains the data item and to which you have access. You may want to use the FORM *data item name* command in session mode before you prepare your job to determine the order of the data sets that include the data item. You can then decide what to include in the data set list.

QUERY will inform you of which data set was chosen, if there is any ambiguity, with this message:

data item name IS A MEMBER OF THESE SETS:
data set name,data set name, ...
data set name USED

Automatic Data Set List Additions

If a FIND or LIST command is executed which contains an unqualified data item name (i.e., a data item without a preceding data set name), QUERY automatically adds the name of the accessed data set to the data set list. This occurs whether the named data item appears in more than one data set or not.

To avoid any ambiguity when the name data item appears in multiple data sets, you can do one of two things:

- Use fully-qualified data item names in all commands.
- Always reset the data set list with the DATA-SETS= command prior to entering a command using a data item which appears in multiple sets.

Examples

Example 1

This example illustrates the DATA-SETS= command. Data set names are entered into the data set list. When the DEFINE command is issued, you can see which sets are currently in the list.

```
>S=INVENTORY,SALES
>DEFINE
DATA-BASE = ORDERS.PUB.SYS
DATA-BASE = >>RETURN
PASSWORD = **********
PASSWORD = >>RETURN
```

```
MODE = 1
MODE = >>RETURN
DATA-SETS = INVENTORY,SALES
DATA-SETS = >>RETURN
PROC-FILE  = >>RETURN
OUTPUT = TERM
OUTPUT = >>RETURN
```

Example 2

This example illustrates how QUERY uses the data set list, clears it, and enters data set names automatically. The DEFINE command shows that the data set list currently contains the SALES data set. When using the FIND command to search entries with STOCK#=6650D22S, the SALES data set is automatically used since it is in the data set list. The DATA SET= command, followed by no data set names, clears the data set list. When FIND STOCK=6650D22S is used again, QUERY prompts for the data set to be used. The DEFINE command shows that the data set INVENTORY was automatically added to the data set list. When FIND STOCK#=7391Z22F is entered, the INVENTORY data set is used. However, in the next FIND command, the data set SALES is specified. In this case, the data set list is not used.

```
>DEFINE
DATA-BASE = ORDERS.PUB.SYS
DATA BASE = >>RETURN
PASSWORD = **********
PASSWORD = >>RETURN
MODE = 1
MODE = >>RETURN
DATA-SETS = SALES
DATA-SETS = >>RETURN
PROC-FILE  = MANPROC.IMAGE.DATAMGT
PROC-FILE  = >>RETURN
OUTPUT = TERM
OUTPUT = >>RETURN
>FIND STOCK#=6650D22S
2 ENTRIES QUALIFIED
>DATA-SETS=
>FIND STOCK#=6650D22S
STOCK#      IS A MEMBER OF THESE SETS:
PRODUCT,SALES,INVENTORY
WHICH SET DO YOU WISH TO USE?
>>INVENTORY
5  ENTRIES QUALIFIED
>DEFINE
DATA-BASE = ORDERS.PUB.SYS
DATA-BASE = >>RETURN
PASSWORD = **********
PASSWORD = >>RETURN
MODE = 1
MODE = >>RETURN
DATA-SETS = INVENTORY
DATA-SETS = >>RETURN
PROC-FILE  = MANPROC.IMAGE.DATAMGT
PROC-FILE  = >>RETURN
OUTPUT = TERM
OUTPUT = >>RETURN
>FIND STOCK#=7391Z22F
2  ENTRIES QUALIFIED
>FIND SALES.STOCK#= 7391Z22F
8  ENTRIES QUALIFIED
>
```

DBLIST=

Informs QUERY which data base to access for multiple data set retrieval and reporting.

Syntax

DBLIST= [*data base list*]

For example:

```
>DBLIST=JOBS,SALES

>DBLIST=
```

Parameter

data base list is a list of data bases opened with the DEFINE, DATA-BASE=, and/or MULTIDB commands. Each data base name must be separated from the next with a comma. If *data base list* is not included, the data base list is cleared.

Discussion

In IMAGE, the same name can be used for different data items in a different data base. You can use a fully-qualified data item name to tell QUERY which data item you want to access. The form is: *data base name: data set name. data item name*. However, if you are referring to several data items which appear in multiple data bases, the command could become quite lengthy. The DBLIST= command can save you some effort.

The DBLIST= command defines a list of one or more data bases. When you use a JOIN, MULTIFIND, SUBSET, or REPORT command after a MULTIFIND command, and QUERY encounters a data item name which appears in more than one data base but is not fully-qualified, it checks the data base list to resolve the ambiguity.

Data Base Selection Rules

Session Mode

If you reference a data item appearing in more than one data base, QUERY resolves which data base to use according to the following rule:

If one (and only one) of the data bases containing the data item appears in the data base list, QUERY automatically uses that data base. Otherwise QUERY prompts you to supply the desired data base with the following message:

data item name IS A MEMBER OF THESE DATA BASES:
data base name,data base name,...
WHICH DATA BASE DO YOU WISH TO USE?

>>*data base name*

You must type the name of the data base you want to access. If the name you provide does not match the names listed, QUERY repeats the prompt. If you decide you do not want to

access the listed data bases, you may abort the command by pressing RETURN instead of entering a data base name. You will be prompted for another command with >.

Job Mode

In job mode, QUERY cannot prompt you for the desired data base in the event there is ambiguity. The data base to be accessed is chosen according to the following rules:

- If exactly one data base containing the data item appears in the data base list, QUERY uses that data base.
- If more than one of the data bases containing the data item appears in the data base list, QUERY uses the data base mentioned last in the list. You can check to see which data base is last by using the SHOW DBLIST command.
- If the data base containing the data set or data item is not included in the data base list, and the only data base open is the primary data base, QUERY uses that data base. If additional data bases have been opened the the MULTIDB command, QUERY uses the data base which was opened last that contains the data set or data item.

If there was any ambiguity, QUERY informs you which data base was used with the following message:

data item name IS A MEMBER OF THESE DATA BASES:
data base name,data base name,...
data base name USED

Automatic Data Base List Additions

If a JOIN, MULTIFIND, SUBSET, or REPORT command which contains an unqualified data item name is executed following a MULTIFIND command, QUERY automatically adds the accessed data base name to the data base list. This occurs whether or not the named data item occurs in more than one data base.

To avoid ambiguity when the data item appears in more than one data base, you can either use fully-qualified data item names in all commands or always reset the data base list (using the DBLIST= command) prior to entering a command which uses a data item that appears in more than one data base.

Example

This example illustrates how QUERY uses the data base list, clears it, and enters data base names automatically.

```
>SHOW DBLIST
ALBUMS
>JOIN JAZZ.ARTIST TO TAPES:JAZZ.ARTIST
>MULTIFIND JAZZ.LABEL="MMM"
USING SERIAL READ
2 COMPOUND ENTRIES QUALIFIED
>DBLIST=
>REPORT
>>OUT=LP
>>D1,JAZZ.LABEL,30
JAZZ            IS A MEMBER OF THESE DATA BASES:
ALBUMS,TAPES
WHICH DATA BASE DO YOU WISH TO USE?
>>TAPES
```

```
>>END
>SHOW DBLIST
TAPES
>MULTIFIND JAZZ.YEAR=70
USING SERIAL READ
3 COMPOUND ENTRIES QUALIFIED
>
```

The SHOW DBLIST command shows that ALBUMS is the only data base in the data base list. When the JOIN command is entered, ALBUMS is automatically used for the first occurrence of the JAZZ data set. In the second occurrence, the data base list is not used since a fully-qualified data item name is specified. The DBLIST= command clears the data base list. Since the data base list is cleared when the REPORT command is entered, QUERY prompts for the data base to be used. TAPES is automatically added to the data base list. When the next MULTIFIND command is entered, QUERY automatically uses the TAPES data base because the data item name is not fully-qualified.

DEFINE

Lists the status of all of the environment commands for the primary data base and enables you to change the environment specifications.

Syntax

```
DEF[INE]
```

Discussion

When you enter the DEFINE command for the primary data base in either session or job mode, QUERY lists the state of each of the environment commands and prompts you for changes. The primary data base is the data base opened by the DEFINE or DATA-BASE= command. After the prompt, you can enter a new parameter for the command, or press return to maintain the current value.

When DEFINE is entered in job mode, you must anticipate the order of QUERY prompts and enter the new values in the proper order on the records immediately following the DEFINE command. A blank record indicates no change in value and is treated like RETURN in session mode.

The DEFINE command prompts are shown below:

```
DATA-BASE = >>
PASSWORD = >>
MODE = >>
DATA-SETS = >>
PROC-FILE  = >>
OUTPUT = TERM
OUTPUT = >>
```

When you first use the DEFINE command after initializing QUERY execution, none of the command settings will be listed except OUTPUT=TERM. If you do not want to define a particular environment parameter, respond with RETURN. If you are only doing procedure maintenance, you need not specify a data base. On the other hand, if you are not using procedures you do not have to define a Proc-file.

If you are signed on as the creator and enter a semicolon in place of the data base password, you will be given read and write access to all data items and data sets in the specified data base. This is true even if no passwords are defined for the data base.

If you enter an invalid password, you will be assigned a user class of zero, which allows you read and/or write access to some or all of the data sets and data items in the specified data base. When this happens, the following message is returned:

```
PASSWORD DOES NOT MATCH ANY DEFINED FOR THE SPECIFIED DATA BASE;
USER CLASS ZERO (0) WAS ASSIGNED
```

If user class zero does not allow you access to any data sets and/or data items in the data base, the following message is returned:

```
BAD PASSWORD
```

If you are not sure that you will have access to the data items you need, you can change the password with the DEFINE, DATA-BASE=, or PASSWORD= command.

Refer to the command description of each prompt for more information about the purpose of each environment specification.

Examples

Example 1

```
>DEFINE
DATA-BASE = >>ORDERS
PASSWORD = >>CLERK
MODE = >>5
DATA-SETS = >>RETURN
PROC-FILE  = >>MANPROC
OUTPUT = TERM
OUTPUT = >>RETURN
>
```

The DEFINE command can be used to set up the environment of the primary data base for your QUERY session. QUERY prompts for the data base name, password, mode, data set list, and procedure file name. The current output device is the terminal because RETURN is pressed in response to the prompt.

Example 2

```
>DEFINE
      DATA-BASE = ORDERS
      DATA-BASE = >>RETURN
      PASSWORD = **********
      PASSWORD = >>RETURN
      MODE = 5
      MODE = >>RETURN
      DATA-SETS = >>RETURN
      PROC-FILE  = MANPROC
      PROC-FILE  = >>CANNED
      OUTPUT = TERM
      OUTPUT = >>RETURN
      >
```

The DEFINE command is also useful for listing the current environment of the primary data base and changing it. Once the environment for the primary data base has been defined, QUERY prints the current setting for each environment command and allows you to change it or press return to leave it as is. In the example above, the Proc-file is changed from MANPROC to CANNED.

DESTROY

Deletes a procedure from the current Proc-file.

Syntax

DE[STROY] *procedure name*

For example:

```
>DESTROY REP23
```

Where*procedure name* = REP23

Parameter

procedure name is the name of a procedure stored in the current Proc-file.

Discussion

This command deletes a procedure from the current Proc-file. If the Proc-file has not been declared, or if the named procedure does not exist in the Proc-file, QUERY informs you and prompts you for another command. If the procedure does exist in the Proc-file, it is deleted.

Example

```
>PROC-FILE  =MANPROC
>DISPLAY LIST

FIND1       FIND2       UPD1        UPD2        REP4        REP5
>DESTROY UPD2
>D LIST

FIND1       FIND2       UPD1        REP4        REP5
>
```

DISPLAY

Lists a procedure which is stored in the current Proc-file.

Syntax

D[ISPLAY] *procedure name* [*m* [,*n*] / ,*file name*]

For example:

```
>D PROCA,2,4
```

Where *procedure name* = PROCA, *m* = 2, *n* = 4

```
>DISPLAY GETIT
```

Where *procedure name* = GETIT

```
>DISPLAY REP4,FILEB
```

Where *procedure name* = REP4, *file name* = FILEB

Parameters

procedure name	is the name of a procedure in the current Proc-file. Refer to Table 2-2 for a list of commands which can be used in procedures.
m and *n*	are the first and last line of the procedure (respectively) to be displayed. *m* must be an integer less than or equal to *n* and greater than or equal to 1.
file name	is the name of an MPE ASCII file into which the displayed procedure is to be written.

Discussion

When you use this command, QUERY searches the current Proc-file for the named procedure. If the procedure does not exist or if no Proc-file has been declared, you are informed and prompted for another command.

If the procedure does exist, it is listed with line numbers for reference when editing the procedure with the ALTER command. If line numbers (*m,n*) are included in the DISPLAY command, only those lines specified will be listed. If only *m* is included, the procedure will be listed from line *m* to the end of the procedure.

Displaying to a File

If a *file name* is included in this command, QUERY writes the procedure statements (excluding line numbers and header which appear as part of the procedure listing) to the specified file. The entire procedure must be transferred. Any existing information in the file is overwritten.

If you have access to save files and *file name* does not exist in your log-on group, QUERY creates a new file using *file name* as the formal designator (name of the file). The file size will be 200 records. The maximum Proc-file record size is 125 words. The following message is printed when this situation occurs.

```
FILE DOES NOT EXIST, BEING CREATED
```

Example

```
>DISPLAY FIND1,FINDSET
>DISPLAY FIND1

PROCEDURE: FIND1

001   FIND
002   CUSTOMER.ACCOUNT IS ""
003   END
```

In the example above, the FIND1 procedure is written to file FINDSET. Then FIND1 is listed on the terminal.

```
>OUT=LP
>D FINDCH
>OUT=TERM
>D FINDCH


PROCEDURE: FINDCH

001   FIND CHAIN
002   CUSTOMER.ACCOUNT,SALES.ACCOUNT
003   IS "" END
```

When OUTPUT=LP, the list is printed on the QSLIST device. When OUTPUT=TERM, the list is printed on the terminal.

```
>D FINDCH,2


PROCEDURE: FINCH

002   CUSTOMER.ACCOUNT,SALES.ACCOUNT
003   IS "" END
```

A single line or a range of lines can be displayed.

DISPLAY LIST

Lists the names of all procedures in the current Proc-file.

Syntax

D[ISPLAY]LIST

Discussion

If you enter this command, QUERY prints the names of all procedures in the current Proc-file. If no Proc-file has been declared, you are informed and prompted for another command.

If you have specified OUTPUT=LP (refer to the OUTPUT= command), the DISPLAY LIST command prints information on both the QSLIST device and the terminal in session mode. In this case, the procedure listing includes a header containing the procedure file name and the date you displayed the procedure names.

Example

```
>PROC-FILE  =MANPROC
>OUTPUT=LP
>DISPLAY LIST

FIND1       FIND2       UPD1       UPD2       REP4       REP5
```

If OUTPUT=LP, DISPLAY LIST lists on both the terminal and the QSLIST device. Below is a listing from the QSLIST device:

```
      PROCEDURE FILE: MANPROC.IMAGE.DATAMGT       TUE, JAN 7, 1987

      FIND1       FIND2       UPD1       UPD2       REP4       REP5
```

EXIT

Terminates QUERY execution.

Syntax

```
E[XIT]
```

Discussion

You can enter the EXIT command whenever QUERY prompts for a command. QUERY execution terminates and control is returned to the operating system. The operating system then prompts you for a command with the colon prompt character.

In job mode, you must use an EXIT command to terminate a set of QUERY commands. The EXIT record should be followed by an MPE command (usually :EOJ). Refer to Section 2 for further information on using QUERY in job mode.

Example

```
>EXIT

END OF PROGRAM
:BYE

CPU=9.  CONNECT=6.  MON, MAR 23, 1987, 10:31 AM
```

FIND

Locates entries in a single data set.

Syntax

F[IND][#LIMIT=*i*;] { *relation* / *item identifier* M[ATCHING] "*pattern*" }
[{ AND / OR } { *relation* / *item identifier* M[ATCHING] "*pattern*" }] ... [END]

For example:

```
>FIND BADGE# IS "09.18"
```

Where *relation* = `BADGE# IS "09.18"`

```
>FIND L-NAME IS MARTENSEN AND F-NAME IS SALLY
```

Where *relation* = `L-NAME IS MARTENSEN AND F-NAME IS SALLY`

```
>F STATE M "C?"
```

Where *item identifier* = `STATE` and *pattern* = `"C?"`

Parameters

i is an integer specifying the maximum number of qualifying entries you want to retrieve. *i* must be > 0. If you specify a negative number, QUERY ignores your input. When the #LIMIT = parameter is specified, only the first *i* qualifying entries are placed in the select file. If fewer than *i* qualifying entries exist, then all entries that qualify are put into the select file.

item identifier takes the form:

[*data base name:*] [*data set name.*] *data item name*

[(*subscript*)]

data base name is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of a data set in a currently open data base. If a data base is specified, the data set must belong to that data base.

data item name is either the name of a simple data item, or the name of a compound data item with an optional (*subscript*) parameter. If a data set name is used, the data item must belong to the specified data set. For matching, the data item must be type X or U.

subscript is a number to indicate which sub-item you want to locate. *Subscript* is entered with parenthesis, and must be an integer >= 1 and <= the number of sub-items defined for the compound data item. QUERY defaults to the first sub-item if no *subscript* is specified.

`MATCHING` allows you to retrieve data based on the comparison of data item values with a specified *pattern*.

"pattern" is defined by pattern matching specifications. *Pattern* must be enclosed in quotation marks.

relation takes the form:

[*data base name:*] [*data set name.*] *data item name*

[(*subscript*)] *relop "value"* [,*"value"*]...

relop is a relational operator as shown in Table 3-1.

value is the data item value. It must be the same type and within the same value range as the data item named in the *relation. Value* need not be enclosed in quotation marks unless the value contains special characters. Values not in quotation marks will be upshifted. For example, California is changed to CALIFORNIA before it is compared to data item values in the data base. However, data items of type X are not upshifted if no quotation marks are used. *Value* must be an exact match for character type data items (type U and X).

Null values will cause QUERY to prompt you for the *values* that you want to retrieve. Refer to "Using Null Values" under the FIND command.

END must be included in a procedure.

Methods of Retrieval

If a FIND command is entered, QUERY will use either a keyed (indexed) or a serial method of retrieval. A keyed retrieval occurs when master or detail data set search keys can be used to locate entries. A serial read occurs when the data set requires serial scanning without the benefit of search keys.

In some cases, a serial method of retrieval may take a great deal of time. If you decide you do not want to wait for completion, you can abort a search by entering CONTROL Y. QUERY will print the number of qualifying entries and ask you if you want to continue the search. A "NO" reply will store the record addresses of retrieved entries in a select file and discontinue retrieval of additional entries. Search time can also be controlled with the #LIMIT = parameter, which specifies the maximum number of qualifying entries you want to retrieve.

Table 3-1. FIND Command Relational Operators

OPERATOR	MEANING
= IS IE EQ	is equal to (Multiple *values* may be used with these operators.)
# <> ISNOT INE NE	is not equal to (Multiple *values* may be used with these operators.)
< ILT LT	is less than
>= INLT GE	is not less than (is greater than or equal to)
> IGT GT	is greater than
<= INGT LE	is not greater than (is less than or equal to)
IB $value_1$, $value_2$	is between (and including) $value_1$ and $value_2$
	Note: The operators <>, <=, and >= cannot have any intervening spaces (embedded blanks).

U and X Type Values

When entering values for X and U type data items, the values must appear exactly as the data was originally entered. For example, if the data item STREET-ADDRESS has a value with three spaces between the street number and name, you must enter those spaces or QUERY will not find the item. Leading blanks must also be entered if they appear in the item. Since blanks or spaces are special characters, all such values must be enclosed in quotation marks.

Logical Connectors

To make more than one comparison for each entry selected, you connect relations with the logical connectors AND or OR. The AND connector instructs QUERY to select only those entries whose data item values satisfy the relations on both sides of the AND. The OR connector indicates that the entries are selected if one (or both) of the two relations on either side of the OR is satisfied

Both types of logical connectors can appear in a FIND command. All relations connected with AND are examined as if they were surrounded by parenthesis. Any relation or set of relations separated from others by OR are compared to the data entry and the entry is selected if the relations are true. For example, the command:

```
>FIND A = 3 AND B = 4 OR C IGT 9 END
```

locates all entries with both A equal to 3 and B equal to 4 as well as all entries containing a C value greater than 9. The command:

```
>FIND A = 3 OR A = 2 AND B = 5 AND C = 8 OR B = 9 END
```

locates all entries with either A equal to 3 or A equal to 2 and B equal to 5 *and* C equal to 8, or entries with B equal to 9.

Parentheses cannot be used in a command, but you can create constructs which act as parentheses and force an OR comparison to take precedence over an AND comparison. For example, if C*n* stands for a relation:

(C_1 and C_2) and C_3

is represented as:

C_1 and C_3 or C_2 and C_3

Up to 50 logical connectors may be used in one FIND command.

Compound Items

A compound data item is an item that occurs more than once in the same data entry. Each occurrence of the data item is called a sub-item. Each may have a value and any or all sub-items in a data entry can be accessed.

For example, if you have defined a compound item called MONTHLY-SALES with 12 sub-items (1 for each month), a retrieval of all entries with June sales of greater than $2000 could look like this:

```
>FIND MONTHLY-SALES(6) > 2000
```

If no *subscript* is specified, the *subscript* is assumed to be the first sub-item.

Multiple Values

To specify more than one value for the same data item, list the values one after the other, separated by commas. For example, the command:

```
>FIND STATE IS "CALIFORNIA","NEVADA","WASHINGTON"
```

locates the entries with the value of data item STATE equal to either CALIFORNIA, NEVADA, OR WASHINGTON. The above command is equal to:

```
>FIND STATE IS "CALIFORNIA" OR STATE IS "NEVADA" OR STATE IS "WASHINGTON"
```

Multiple values can only be used with the “equal” or “not equal” relational operators.

Using Null Values

The FIND command can prompt you for data item values to be compared with data item values in the data entries of the data set. To do this, you use null data item values in the command. Null values are represented by a pair of quotation marks without any intervening characters or blank spaces (””). When the command is executed, you are prompted to enter a value for each null value in the command. This is useful when the FIND command is stored as a procedure in a Proc-file.

The procedure can be executed using different comparison values without modifying the procedure each time. For example, the following command would prompt you for a value of ACCOUNT with the following message.

```
>FIND SALES.ACCOUNT <> ""
WHAT IS THE VALUE OF - ACCOUNT
>>24536173
USING SERIAL READ
10  ENTRIES QUALIFIED
```

QUERY searches the appropriate data set for the values you specify. The value should be entered *without* the surrounding quotes since all characters entered (including leading blanks, quotes, and other special characters) are significant. Lowercase characters are upshifted unless the data item type is X. The maximum value size is 72 characters.

You are prompted once for each null value in the command. You must supply a value for each null value in the procedure. You can use a known invalid response if you do not want to find another entry. For example:

```
>FIND CUSTOMER.CITY IS "","" 
WHAT IS THE VALUE OF - CITY
>>PETALUMA
WHAT IS THE VALUE OF - CITY
>>X
USING SERIAL READ
1 ENTRIES QUALIFIED.
```

When using null values from a FIND command in an XEQ file, the XEQ parameter NODATA must be used. This parameter indicates that the values to be searched for are not included in the FIND command but must be entered by you. Refer to the XEQ command for more information about using null values.

In session mode, supply the desired value after each prompt message. In job mode, you must anticipate the order of prompts and supply the desired values, one per record, following the FIND command.

Examples

Example 1

To determine whether or not a customer is already in the data set, you can try to find the name.

```
>FIND LAST-NAME IS MARTENSEN
USING SERIAL READ
0  ENTRIES QUALIFIED
```

Example 2

Since ACCOUNT is in more than one data set you can either qualify the data item name or let QUERY prompt you for it. The response, SALES, is automatically entered in the data set list.

```
>FIND CUSTOMER.ACCOUNT EQ 24536173
USING SERIAL READ
1  ENTRIES QUALIFIED
>DATA SETS=
>FIND ACCOUNT EQ 24536173
ACCOUNT            IS A MEMBER OF THESE SETS:
CUSTOMER,SALES
WHICH SET DO YOU WISH TO USE?
>>SALES
3  ENTRIES QUALIFIED
```

Example 3

In the example below, QUERY uses the data set list to determine which of the three data sets containing STOCK# to use. Since SALES was placed in the data set list in the previous example, it is used. However, the data item DESCRIPTION is in the PRODUCT data set, and the FIND command can only refer to one data set so an error results. It is usually best to

clear the data set list if you are unsure about what it contains. PRODUCT was entered in the data set list automatically as a result of the previous command, so QUERY uses PRODUCT and does not prompt for the data set name.

```
>F STOCK# IGT 33333333 AND DESCRIPTION IS "NEHRU JACKET"
RETRIEVAL FROM MORE THAN ONE DATA SET
>S=
>F STOCK# IGT 33333333 AND DESCRIPTION IS "NEHRU JACKET"
STOCK#       IS A MEMBER OF THESE SETS:
PRODUCT, SALES, INVENTORY
WHICH SET DO YOU WISH TO USE?
>>PRODUCT
USING SERIAL READ
0 ENTRIES QUALIFIED
>F STOCK# IGT 33333333 AND DESCRIPTION IS "NEHRU JACKET"
USING SERIAL READ
0  ENTRIES QUALIFIED
```

Example 4

The following error occurs because an ampersand is used to continue the line but there is no space before it or at the beginning of the next line.

```
>F ACCOUNT IGT 55555555 AND STATE IS CA OR ACCOUNT IS 121212121&
>>AND STATE IS MA OR CUSTOMER.STATE IS AZ
ACCOUNT     IS A MEMBER OF THESE SETS:
CUSTOMER,SALES
WHICH SET DO YOU WISH TO USE?
>>CUSTOMER
INVALID NUMERIC DIGIT
```

Example 5

All items which appear in multiple sets must be qualified unless the set name is in the set list. After entries are located, you can use the REPORT command to print the data.

```
>F CUSTOMER.ACCOUNT IGT 55555555 AND STATE IS CA OR ACCOUNT&
>>     IS 121212 AND STATE IS MA OR STATE IS AZ
STATE IS A MEMBER OF THESE SETS
CUSTOMER,SUP-MASTER
WHICH SET DO YOU WISH TO USE?
>>CUSTOMER
6  ENTRIES QUALIFIED
>REPORT D,ACCOUNT,10;D,STATE,15;END


76623455   CA
74001813   CA
87654321   CA
80808080   CA
77765555   CA
99998877   CA
```

Example 6

A simpler technique for determining the data set to be used is to enter it into the data set list with a DATA-SETS= command.

```
>S=CUSTOMER
>F ACCOUNT IGT 55555555 AND STATE IS CA OR ACCOUNT IS 12121212&
>>      AND STATE IS MA OR STATE IS AZ
USING SERIAL READ
6  ENTRIES QUALIFIED
```

Generic Search

A pattern consists of a series of the following special characters that indicate the type of data that can be entered in that position:

```
text
      a    upper or lowercase alphabetic character (A-Z, a-z)
      u    uppercase alphabetic character (A-Z)
      l    lowercase alphabetic character (a-z)
      b    blank
      d    digit (0-9)
      ?    any character
```

Note: A pattern must specify the maximum number of characters allowed for the data item length. Therefore, to find data item values of varying lengths, you must specify that the rest of the data item value is filled with blanks. Refer to the discussion of repetition for more information.

The beginning of the pattern is defined by a beginning quotation mark and the end is defined by an ending quotation mark. QUERY does not read spaces as blanks; therefore, you can leave spaces inside a pattern to allow for better readability.

The matching pattern can include specific characters in addition to the types listed above. For example:

```
MATCHING "Aaa-dddd"
```

The pattern above means that the value must start with the letter "A" followed by any two upper or lowercase letters, followed by a hyphen and any four digits. For example, the values "Acs-1234" and "AAA-9999" are acceptable, but the values "Bcs-1223" and "A12-345" are not acceptable.

Transparency

A special operator can be used to indicate that a pattern is to be used as an actual value. For example, suppose you want the lowercase letter "a" to be an exact value in the pattern, you can do this by preceding it, or any of the other special characters, with an exclamation point (!). For example:

`MATCHING "!ad"`	Value must start with the letter "a" followed by one digit.

The exclamation point (called the transparency operator) is also used to allow inclusion of any of the pattern operators listed below and described in Table 3-2.

```
!     transparency
,     choice
:     range
{ }   grouping
[ ]   optional
+     repetition  (1 or more)
*     repetition  (0 or more)
```

Choice

You can indicate a selection of acceptable patterns as part of the MATCHING pattern. Each possible choice is separated by a comma. For example:

`MATCHING "ABCD,DEFG,dddd"`	The values "ABCD", "DEFG", or any four digits are acceptable.

Range

A range of acceptable characters for a single character position can be indicated with the colon. All characters within the range are acceptable. This acts as shorthand for listing a series of single characters in ASCII sequence. For example:

`MATCHING "Cb:Jb"`	This pattern would accept the values C, D, E, F, G, H, I, J in a two-character field.
`MATCHING "10:15"`	The values 10, 11, 12, 13, 14, and 15 are accepted by this pattern.
`MATCHING "!a?:f?"`	Since "a" is a special character, it is preceded by an exclamation point (!). Other characters in the range (except the special character ?) are implicitly preceded by this operator. This pattern is equivalent to: `MATCHING "!a?,!b?,!c?,!d?,!e?,!f?"`.

Grouping and Optional

You can group pattern specifications by enclosing the pattern in braces { } or brackets [ ]. Braces indicate that data must correspond to one item in the group. Brackets make the pattern optional, indicating that data can correspond to one or none of the items in the pattern. For example:

`MATCHING "{AAA,BBB,CCC} ddd"`	One of the choices within the braces must be matched. The values "AAA123" and "BBB999" and "CCC562", among others, are acceptable matches for this pattern.
`MATCHING "[A,B,C] ddd [b]"`	The choices within the brackets can be omitted, or one can be matched. For example, "A345", "C567", "B441", and "123" are acceptable matches.
`MATCHING "[u,d] !+ [1:5]"`	Accepts such values as "A+", "3+", and "+5".
`MATCHING "{[-,dd] dd [b,d]}"`	Accepts such values as "-125" or "2345" or "500" or "-10".

Repetition

Repetition of any character or sets of characters can be indicated by an asterisk (*) or by a plus sign (+) following any pattern character or pattern group within braces { }. A plus sign (+) means that at least one occurrence of the pattern is required for the match. An asterisk

(*) means that zero or more occurrences can be matched. (These repetition indicators cannot follow items enclosed within brackets [ ].) For example:

`MATCHING "d+b*"`	The plus sign indicates repetition of the digit, with at least one occurrence required for the match. Thus, "2" or "745227" or "55" are acceptable, but a blank is not.
`MATCHING "Xd+b*"`	This pattern accepts the letter X followed by one or more digits. "X1" or "X12323" and so forth are acceptable, but not "X".
`MATCHING "M {A,C,d}+"`	A plus sign after the braces indicates repetition of any item within the braces, in any order. Some acceptable values are "MAC1", "MCCC", or "M123".
`MATCHING "d*b*"`	The asterisk indicates optional repetition that allows zero or more occurrences of the pattern. Thus, the digit can be omitted, or repeated any number of times. Nothing or "3" or "123456" are all acceptable patterns.
`MATCHING "[d+]b*"`	This pattern is another way of expressing the pattern shown above as d*b*.
`MATCHING "a+"`	Accepts an alphabetic value.
`MATCHING "Xu*b*"`	This pattern accepts "X" alone or followed by any number of uppercase letters. For example, "XABC" or "XX" or "X" are all acceptable.
`MATCHING "M+ {A,D,d}*"`	Any of the enclosed characters can be repeated in any order, or can be omitted. Thus, "MMMM" is acceptable, as are "MAA1", "MCCA", "M222", and so forth.

When you use a pattern to find variable length values such as "1", "12" and "123", you must indicate that blanks following the value fill the rest of the length of the item. For example, if a data item type is X6 and the value contained in the item is "123", there are three blanks following the value. In this case, the pattern "ddd" will not find the value. The pattern must account for the blanks that fill the item length. The pattern "dddb*" will find the value if blanks follow the value. If other characters or unknown characters follow the value, you can use the pattern "ddd?*".

Operator Hierarchy

The pattern operators are evaluated in the following order, where x and y are any patterns.

`Highest`	`!x`	*Transparency*
`|`	`x:y`	*Range*
`|`	`x+ or x*`	*Repetition*
`v`	`xy`	*Concatenation*
`Lowest`	`x,y`	*Choice*

Some further examples of the MATCHING statement are:

`MATCHING "1ddd"`	Accepts an integer between 1000 and 1999. Can also be expressed as "1000:1999".
`MATCHING "[d] [d!:dd] b* [AM,PM]"`	Accepts a time such as "3:00 PM" or "12:00".
`MATCHING "{1:7} {0:7}* b*"`	Accepts a number greater than zero with at least one digit and no leading zeros, such as "2047", or "1", or "74".
`MATCHING "ddd-dd-ddddb"`	Accepts any social security number, such as "044-24-0474".
`MATCHING "[(ddd)] ddd-dddd b*"`	Accepts a phone number with an optional area code.

Table 3-2 summarizes the operators allowed in a MATCHING pattern.

Table 3-2. MATCHING Pattern Operators

OPERATOR	FUNCTION	EXAMPLE
!	Transparency operator allows use of any special MATCHING characters as an element in the pattern.	`MATCHING "!u,!d,!,,!!"` accepts any of the values `"u"`, `"d"`, `","` or `"!"`.
,	Choice of subpatterns, any one of which satisfies the match.	`MATCHING "A,B,dd"` accepts values such as `"A"`, `"B"` and `"22"`.
:	Range of single characters in ascending ASCII order, any one of which satisfies the match.	`MATCHING "2:6"` accepts the values `"2"`, `"3"`, `"4"`, `"5"` or `"6"`.
{ }	Grouping (required) requires one occurrence of any pattern within braces.	`MATCHING "{A,B}dd{%,d}"` accepts `"A223"`, `"B34%"`, `"A795"` and so forth.
[]	Grouping (optional) allows zero or 1 occurrence of any item in a pattern within brackets [ ].	`MATCHING "[A,B]dd[%,d]"` accepts `"24"`, `"A99"`, `"10%"`, `"123"` and so forth.
+	Repetition (required) requires one or more occurrences of a preceding item, or a pattern within braces { }.	`MATCHING "Xd+"` accepts values such as `"X1"`, `"X22"`, `"X3334789"` and so forth, but not `"X"`. `MATCHING "{d,a}+"` accepts values such as `"11"` `"A23"`, `"acb"`, `"33ABC9"`.
*	Repetition (optional) allows zero or more occurrences of a preceding item or a pattern within braces { }.	`MATCHING "Xd*"` accepts values such as `"X"`, `"X1"`, `"X22"` and `"X3334789"`. `MATCHING "{d,a}*"` accepts a null value, or such values as `"11"`, `"A23"`, `"acb"` or `"33ABC9"`.

FIND ALL

Locates all entries in the data set regardless of the value of the data item specified.

Syntax

F[IND]ALL [#LIMIT=*i*;] *item identifier*

For example:

```
>FIND ALL LABOR.BADGE#
```

Where *item identifier*" = LABOR.BADGE#"

```
>F ALL F-NAME
```

Where *item identifier* = F-NAME

Parameters

i is an integer specifying the maximum number of qualifying entries you want to retrieve. *i* must be >= 0. If you specify a negative number, QUERY ignores your input. When the #LIMIT = parameter is specified, only the first qualified entries are placed in the select file. If fewer than *i* qualifying entries exist, then all entries that qualify are placed in the select file.

item identifier takes the form:

[*data base name:*] [*data set name.*] *data item name*

data base name is the name of a data base specified in the last DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of the data set you want to access.

data item name is the name of any data item in that set.

Discussion

If you want to locate all entries in a data set (and use these entries in a report, for example), this is an easy way to do so.

If you do not specify *data set name*, QUERY will check the data set list and follow the same rules as defined when using a FIND command with a single data set. (Refer to the FIND command for more information.)

Example

```
>F ALL CUSTOMER.ACCOUNT
USING SERIAL READ
13 ENTRIES QUALIFIED
>R D,ACCOUNT,8;END


54283540
54283545
10293847
   .
```

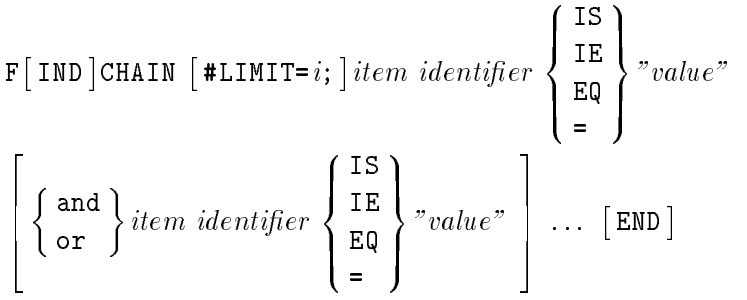
```
        .
        .

24536173
24566356
10034765
>FIND ALL LAST-NAME
USING SERIAL READ
13 ENTRIES QUALIFIED
```

Both FIND commands locate entries in the CUSTOMER data set. The REPORT command prints the value of ACCOUNT for each entry.

FIND CHAIN

Locates data entries from a detail data set and one or more of its corresponding master data sets.

Syntax



For example:

```
>F CHAIN EMPLOYEE.BADGE#,LABOR.BADGE# IS "1234" OR&
>>    EMPLOYEE.BADGE#,LABOR.BADGE# IS "9018"
```

Where *item identifier* = EMPLOYEE.BADGE#,LABOR.BADGE#, and *"value"* = "1234" or "9018"

```
 >FIND CHAIN EMPLOYEE.BADGE#,LABOR.BADGE# IE "1234"

text
Where master set name = EMPLOYEE, master search item = BADGE#,
      detail set name = LABOR, detail search item = BADGE#,
      value = "1234"
```

Parameters

i is an integer specifying the maximum number of qualifying entries you want to retrieve. *i* must be >= 0. If you specify a negative number, QUERY ignores your input. When the #LIMIT = parameter is specified, only the first qualified entries are placed in the select file. If fewer than *i* entries exist, all entries that qualify are placed in the select file.

item identifier takes the form:

master set name. master search item,

detail set name. detail search item

master set name is the name of a master data set in the data base.

master search item is the name of a data item defined as the search item for the master data set.

detail set name is the name of a detail data set that is associated with the master data set previously named.

detail search item is the name of a data item defined as a search item for the detail data set. This search item must provide the link between the detail set and the previously defined master set.

value is a data item value. It must be the same type and within the same value range as the *master search item* and *detail search item* of the data item. *Value* must be enclosed in quotation marks only if it contains special characters or blanks. Null values may be used. Refer to "Using Null Values" under the FIND command. *Value* must be an exact match for character type data items (type U or X).

must be included in a procedure.

END

Discussion

Only one detail data set can be accessed using a FIND CHAIN command, but multiple master sets can be used if they relate to the same detail data set through search items.

The data base administrator can provide you with the information required to use this command, or you can use the FORM command to determine the data item and data set relations.

When you enter FIND CHAIN, QUERY searches the specified master data sets for an entry containing the specified search item value. Then QUERY searches the specified detail data set for entries containing the same search item value. Detail entries with the same search item value are called *chains.* The effect of FIND CHAIN is to locate all the members of a detail chain and the master data entry which constitutes the chain head. For more information on chains, chain heads, and data set relations, consult the *IMAGE Reference Manual.*

When using the FIND CHAIN command:

- Only selection on the basis of equality may be made. Relational operators other than IS, IE, EQ, and = are not allowed.
- Only one value per relational operator is allowed.
- Data items named in the command must always be search items (as defined in the data base).
- Up to 50 logical connectors (AND, OR) can appear in the same command. Refer to the FIND command for more information about logical connectors.
- No more than 16 values can be used in each FIND CHAIN command.

Example

In the example below, CUSTOMER is searched for an entry containing ACCOUNT equal to 76623455, and one entry is found. Then PRODUCT is searched for an entry containing a STOCK# equal to 6550D22S, and one entry is found. Then SALES is searched for entries with both ACCOUNT equal to 76623455 and STOCK# equal to 6550D22S, but no entries are located. CUSTOMER is searched for entries containing ACCOUNT equal to 54283545 and SALES is searched for entries containing ACCOUNT equal to 54283545. One entry is found for each relation. Note that both STOCK# and ACCOUNT are search items.

```
>FIND CHAIN CUSTOMER.ACCOUNT,SALES.ACCOUNT IS 76623455 AND&
>>    PRODUCT.STOCK#,SALES.STOCK# IS 6550D22S OR&
>>    CUSTOMER.ACCOUNT,SALES.ACCOUNT IS 54283545
```

```
4  ENTRIES QUALIFIED
>R ALL

ACCOUNT          =76623455
LAST-NAME        =MCFALL
FIRST-NAME       =JEFFREY
INITIAL          =X
STREET-ADDRESS   =6650 MONTEREY ROAD
CITY             =CARMEL
STATE            =CA
ZIP              =93921
CREDIT-RATING    =3.20000

STOCK#           =6550D22S
DESCRIPTION      =BASEBALL BAT

ACCOUNT          =54283545
LAST-NAME        =MAYFIELD
FIRST-NAME       =WILLIAM
INITIAL          =
STREET-ADDRESS   =39 41ST AVE.
CITY             =PETALUMA
STATE            =CA
ZIP              =10101
CREDIT-RATING    =8.50000

ACCOUNT          =54283545
STOCK#           =4397D13P
QUANTITY         =1
PRICE            =4590
TAX              =276
TOTAL            =0
PURCH-DATE       =121585
DELIV-DATE       =121685
```

FIND procedure

Executes a FIND procedure stored in the current Proc-file.

Syntax

F[IND]*procedure name* [,*character*]

For example:

```
>FIND ACCTS
```

Where *procedure name* = ACCTS

```
>F USERS,X
```

Where *procedure name* = USERS, and *character* = X

Parameters

procedure name	is the name of a FIND command previously stored as a procedure using the CREATE command. The procedure must exist in the current Proc-file specified with the PROC-FILE = command, or in response to the PROC-FILE prompt of the DEFINE or MULTIDB command.
character	is any printable ASCII character. If character is included in the command, the FIND procedure is listed.

Discussion

QUERY searches the current Proc-file and executes the procedure named in the command. If the Proc-file has not been declared, or the procedure does not exist in the Proc-file, or the procedure is incorrect in some way, you are informed by an error message. If *character* is included in the command, QUERY prints the procedure on the standard list device before executing it.

If null data values appear in the procedure, QUERY prompts you for the necessary values. If the retrieval requires a serial search of a data set, the following message is returned:

```
USING SERIAL READ
```

For more information about storing and using FIND procedures, refer to the CREATE command.

Example

```
>FIND FINDACCT
WHAT IS THE VALUE OF - ACCOUNT
>>54283545
2 ENTRIES QUALIFIED
>R ALL,Z


54283545
MAYFIELD
WILLIAM

37 41ST AVE.
```

```
PETALUMA
CA
10101
 8.50000


54283545
4397D13P
1
4590
276
0
121585
122085
```

The FINDACCT procedure contains:

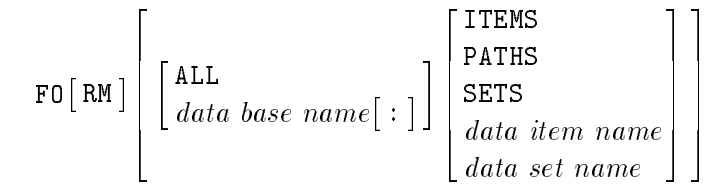
```
FIND CHAIN
CUSTOMER.ACCOUNT,SALES
ACCOUNT
IS "" END
```

In this example, QUERY prompts for the value of ACCOUNT. The REPORT ALL command prints the entry data. The first entry is from the CUSTOMER master data set. The second entry is from the SALES detail data set.

FORM

Lists information about data bases currently being accessed.

Syntax



For example:

```
>FORM

>FO PATHS

>FORM PRODUCT
```

Where *data set name* = PRODUCT

```
>FO ACCOUNT
```

Where *data item name* =ACCOUNT

Parameters

ALL	used alone will display the data sets, data items, and path information for every open data base to which you have access. If used with additional FORM parameters, QUERY will display information designated by these parameters for every open data base.
data base name	is the name of a data base opened with either MULTIDB, DEFINE, or DATA-BASE= commands. If not specified, FORM will use the primary data base currently defined by the DEFINE or DATA-BASE= command. *data base name* may be used to qualify ALL. *data base name* must be followed by a colon when qualifying another parameter.
ITEMS	lists information about each data item in the data base to which you have access.
PATHS	lists the relationship between data sets in the data base to which you currently have access.
SETS	lists information about each data set in the data base to which you have access.
data item name	is the name of a data item in the data base currently being accessed.
data set name	is the name of a data set in the data base currently being accessed.

Discussion

FORM provides information about the currently open data base(s). The information contains only the names of data sets and data items to which you have at least read access. No other data sets and data items are listed. If no data base is currently defined, QUERY issues an error message and prompts you for another command.

If OUTPUT=TERM, the listing is sent to the standard list device for the job or session. If OUTPUT=LP, the listing is sent to the file named QSLIST. Refer to the OUTPUT= command for more information about QSLIST.

Figure 3-1 through Figure 3-6 illustrate the output resulting from each FORM command option. If you enter FORM *name* and *name* refers to both a data set and a data item, the data set information is listed. If a data set or a data item has the name SETS, ITEMS, or PATHS, it is treated as a keyword parameter when used in a FORM command.

```
DATA BASE: ORDERS <----- 1        2 -----> TUE, JAN 7, 1986, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000 <----- 3


ITEMS:

   ACCOUNT <----- 4        J2 <----- 5
   CITY                    X12
   CREDIT-RATING           R2
   DATE                    X6
   FIRST-NAME              X10
   INITIAL                 U2
   LAST-NAME               X16
   PURCH-DATE              X6
   STATE                   X2
   STOCK#                  U8
   STREET-ADDRESS          X26
   TOTAL                   J2
   ZIP                     X6
```

Figure 3-1. FORM ITEMS Output

Discussion 3-1

1. Current data base name.
2. Current date and time.
3. Language of the data base entries.
4. Item name.
5. Item type and length.

```
DATA BASE: ORDERS                                    TUE, MAR 9, 1987, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000


PATH IDENTIFYING INFORMATION


                    ASSOCIATED
MASTER SET NAME     DETAIL SET NAME     SEARCH SET NAME     SORT ITEM NAME

CUSTOMER            SALES               ACCOUNT             PURCH-DATE

DATE-MASTER         SALES               PURCH-DATE


                                                            ASSOCIATED
DETAIL SET NAME     SEARCH ITEM NAME    SORT ITEM NAME      MASTER SET NAME

SALES               ACCOUNT             PURCH-DATE          CUSTOMER
                   !STOCK#
                    PURCH-DATE                              DATE-MASTER
```

Figure 3-2. FORM PATHS Output

Discussion 3-2

The FORM PATHS command lists the detail data sets associated with each master data set and the master data sets associated with each detail data set. It also lists the detail set item which is used as a key (search item name) and the detail set item which is used for sorting (if any).

```
DATA BASE: ORDERS                                    TUE, MAR 9, 1987, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000

                          ITEM                  ENTRY     ENTRY     BLOCKING
SETS:              TYPE   COUNT    CAPACITY     COUNT     LENGTH    FACTOR

   CUSTOMER          M       9        2003        15        41        10
   DATE-MASTER       A       1         211        18         3        22
   SALES             D       4        12012       13        19        14

     |               |       |          |          |         |         |
     1               2       3          4          5         6         7
```

Figure 3-3. FORM SETS Output

Discussion 3-3

1. Data set name.
2. Data set type (M=Master, A=Automatic, D=Detail).
3. Number of items in each data set entry.
4. Maximum number of entries each set can contain.
5. Number of entries currently stored in each data set.

6. Number of computer words per entry.
7. Maximum number of entries a block can contain.

```
DATA BASE: ORDERS                                  TUE, MAR 9, 1987, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000


ITEM NAME:

   ACCOUNT                 J2 <-----  1

   IS A MEMBER OF THESE SETS:
      CUSTOMER             <-----  2
      SALES
```

Figure 3-4. FORM *data item name* **Output**

Discussion 3-4

1. Data item type.
2. Data set names.

```
DATA BASE: ORDERS                                   TUE, MAR 9, 1987, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000


SET NAME:
   CUSTOMER,MANUAL <-----  1

      ITEMS:

         ACCOUNT,  <-----  2     J2    3 -----> <<KEY ITEM>>
         LAST-NAME,               X16
         FIRST-NAME,              X10
         INITIAL,                 U2
         STREET ADDRESS,          X26
         CITY,                    X12
         STATE,                   X2
         ZIP,                     X6
         CREDIT-RATING,           R2

      CAPACITY: 2003 <-----  4    ENTRIES 15  <-----  5
```

Figure 3-5. FORM *data set name* **Output**

Discussion 3-5

1. CUSTOMER is a manual master data set.
2. Data items in the CUSTOMER data set.
3. ACCOUNT is a key item linked to a detail data set.

4. Maximum number of entries CUSTOMER can contain.
5. Number of entries in the data set.

Discussion 3-6 (figure follows)

1. ACCOUNT is a key which links the detail data set SALES to the master data set CUSTOMER. The detail search item is is also named ACCOUNT.
2. DATE is a key item in DATE-MASTER linked to the SALES data set through the search item PURCH-DATE.
3. PURCH-DATE is also used for sorting.
4. The SALES data set can contain at most 12012 entries.
5. Data set relations.

```
DATA BASE: ORDERS                                  TUE, MAR 9, 1987, 11:29 AM

DATA BASE LANGUAGE ATTRIBUTE: NATIVE-3000

SET NAME:
   CUSTOMER,MANUAL
      ITEMS:
         ACCOUNT,               J2          1 -----> <<KEY ITEM>>
         LAST-NAME,             X16
         FIRST-NAME,            X10
         INITIAL,               U2
         STREET ADDRESS,        X26
         CITY,                  X12
         STATE,                 X2
         ZIP,                   X6
         CREDIT-RATING,         R2
      CAPACITY: 2003             ENTRIES 15

SET NAME:
   DATE-MASTER,AUTOMATIC
      ITEMS:
         DATE,                  X6          2 -----> <<KEY ITEM>>
      CAPACITY: 211              ENTRIES: 18

SET NAME:
   SALES,DETAIL
      ITEMS:
         ACCOUNT,               J2          1 -----> <<SEARCH ITEM>>
         STOCK#,                U8                   <<SEARCH ITEM>>
         TOTAL,                 J2
         PURCH-DATE,            X6     2 & 3 -----> <<SEARCH ITEM, SORT ITEM>>
      CAPACITY: 12012 <-----4          ENTRIES: 13

PATH IDENTIFYING INFORMATION <-----5

                  ASSOCIATED
MASTER SET NAME   DETAIL SET NAME     SEARCH SET NAME    SORT ITEM NAME

CUSTOMER          SALES               ACCOUNT            PURCH-DATE

DATE-MASTER       SALES               PURCH-DATE

                                                         ASSOCIATED
DETAIL SET NAME   SEARCH ITEM NAME    SORT ITEM NAME     MASTER SET NAME

SALES             ACCOUNT             PURCH-DATE         CUSTOMER
                 !STOCK#
                  PURCH-DATE                            DATE-MASTER
```

Figure 3-6. FORM Output

HELP

Lists information about the function, format, and parameters of QUERY commands.

Syntax

H[ELP][*command name* [FU[NCTION]][FO[RMAT][PA[RAMETERS]]]]

For example:

```
>HELP

>HELP FORM FUNCTION
```

Where *command name* = FORM

```
>H DEFINE
```

Where *command name* = DEFINE

Parameters

command name	consists of any QUERY command name such as DISPLAY or RENAME. You may use the abbreviation of the command.
FUNCTION	indicates that only the function of the command should be printed.
FORMAT	indicates that only the format of the command should be printed.
PARAMETERS	indicates that only the parameters of the command should be printed.

Discussion

The HELP command provides a convenient on-line reference for QUERY commands. If you enter HELP, you receive a list of the QUERY commands followed by a brief description of the function of each command.

If you enter HELP *command name*, you receive information about the format, function, and parameters of the command, if any.

If you enter HELP *command name* followed by one or more of the parameters FUNCTION, FORMAT, or PARAMETERS, QUERY provides only the information you request.

HELP output is listed on the standard list device, unless OUTPUT=LP has been specified. In that case, the output is directed to the device equated to the QSLIST file. (Refer to the OUTPUT command for more information.)

In order to allow you plenty of time to read the command descriptions when using HELP in session mode, QUERY prints the following message after listing several lines.

```
**PRESS ANY KEY TO CONTINUE
```

When you finish reading the current display, press any terminal key and the listing will continue. You can use (CONTROL) Y to end the command description.

Examples

Example 1

You can abbreviate the HELP command and the command you are asking about.

```
>H DATA-BASE

FUNCTION -
   SPECIFY THE PRIMARY DATA BASE

FORMAT -
   DATA-BASE= DATA BASE NAME

PARAMETERS -
   DATA BASE NAME - NAME OF

(CONTROL) Y
< CONTROL Y >
```

Example 2

The order of FUNCTION, PARAMETERS, and FORMAT parameters can vary.

```
>HELP OUTPUT PA FO

PARAMETERS -
   TERM - INDICATES THAT OUTPUT SHOULD BE SENT
          TO THE DEVICE SPECIFIED AS $STDLIST
   LP   - INDICATES THAT OUTPUT SHOULD BE SENT
          TO THE FORMAL FILE DESIGNATOR 'QSLIST'

FORMAT -
   OUTPUT= TERM/LP
```

JOIN

Defines the compound data set used for multiple data set retrieval.

Syntax

J[OIN] *data item equivalence* [, *data item equivalence*] ...
[; *data set equivalence* [, *data set equivalence*] ...] [END]

Parameters

data item equivalence has the form:

[*data base name:*] *data set name. data item name*

[(*subscript*)] [@] TO [@] [*data base name:*]

data set name. data item name [(*subscript*)]

data base name is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of a data set that is to be included in the compound data set being specified. If *data base name* is used, the data set must belong to that data base.

data item name is the name of the data item that links the joined data sets together. The data item must belong to the specified data set.

subscript is a number used to indicate which sub-item you want to join. *Subscript* must be an integer >= 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

@ means "preserve all values of" the data set associated with the @ sign. This parameter is explained in detail later in "Using the @ Parameter".

data set equivalence has the form:

dummy data set name = data set name

dummy data set name is a temporary name for the data set named on the right side of the data set equivalence. A dummy data set cannot be a legal data set name in any open data base.

END may be used to end a JOIN command.

Discussion

The JOIN command allows the retrieval and reporting of data item values from multiple data sets by creating a logical relation between data sets. The relation is established by equating a data item from one set with a data item from a second data set. These data item equivalences, of which QUERY allows up to 52, define a compound data set.

Once you have defined a compound data set with the JOIN command, you must enter a MULTIFIND or a MULTIFIND ALL command to actually create the compound data set and access the desired items. Refer to the MULTIFIND or MULTIFIND ALL commands in this manual for details on their operation. You can only report from those data sets used in the JOIN command.

A JOIN command will remain in effect until another JOIN command, valid or invalid, is entered. Each subsequent JOIN command also clears the previous select file. For example, suppose you want to make a sales report that shows information from four data sets: the name of the product, the quantity sold of that product, the quantity on hand, and the date of sale. To access all this information, define your data base with a DEFINE, DATA BASE=, or MULTIDB command and enter the JOIN command.

```
>JOIN PRODUCT.STOCK# TO SALES.STOCK#,&
>>    SALES.STOCK# TO INVENTORY.STOCK#
>MULTIFIND PRODUCT.DESCRIPTION = NAIL
USING SERIAL READ
2 COMPOUND ENTRIES QUALIFIED
```

In order for the JOIN command to be valid, all data sets used in the JOIN command must be logically connected to one another so that each data set is traceable to every other data set through the data item equivalences. If logical connections cannot be traced between data sets, the following message is returned and the JOIN does not take place.

```
LOGICAL CONNECTIONS ARE
INCOMPLETE, COMPOUND
DATA SET CANNOT BE GENERATED
```

In the example above, the data set SALES is common to both PRODUCT and INVENTORY, and a data equivalence is formed with the data item STOCK#. However, the data items that link the data sets need not have the same name or be of the same data type.

The MULTIFIND PRODUCT.DESCRIPTION=NAIL command created the compound data set from which retrieved entries can be reported. To produce a sales report for the NAIL product, you can enter the REPORT command with the desired header and detail statements from your terminal or you can use a procedure file. The REPORT command is shown on the next page.

```
>REPORT SALESPROC,A

REPORT
H1,"Report on Sales",43,SPACE A2
H2,"Stock",7
H2,"Product",24
H2,"Quantity",39
H2,"Quantity",56
H2,"Date",70
H3,"No.",7
H3,"Description",24
H3,"Sold",39
```

```
H3,"in Stock",56
H3,"Sold",70,SPACE A2
D1,PRODUCT.STOCK#,6
D1,PRODUCT.DESCRIPTION,24
D1,SALES.QUANTITY,39
D1,INVENTORY.ONHANDQTY,56
D1,SALES.PURCH-DATE,70,E1
E1,"XX/XX/XX"
END
```

```
                          Report on Sales

Stock          Product       Quantity         Quantity          Date
  No.      Description           Sold        In Stock          Sold


  50              NAIL            130            1200      85/01/12
  50              NAIL              5            1200      85/01/12
```

The compound data set was formed by matching entries from the data sets PRODUCT, INVENTORY, and SALES that have equal values for connecting data items named in the data item equivalence. In this example, STOCK# is the only equated data item for all data sets.

If a value for the data item named in the data item equivalence occurs more than once in one of the data sets named in the equivalence, and at least once in the other data set named, this entry will be repeated in the compound data set. Given the following data sets and a data item equivalence using upper-limit and price, the name JOHN SMITH will appear twice in the compound data set.

```
          CLIENT                          SELLER
------------------------------      ------------------------------
     NAME     UPPER LIMIT $              NAME         PRICE
------------------------------      ------------------------------
  JOHN SMITH     $1000                 MARY SMITH      $1000
                                     GEORGE BROWN      $1000
------------------------------      ------------------------------
```

```
>JOIN CLIENT.UPPER-LIMIT TO SELLER.ASKING-PRICE
>MULTIFIND ALL
2 COMPOUND ENTRIES QUALIFIED
>REPORT ALL

SALES:  CLIENT
NAME            =JOHN SMITH
UPPER-LIMIT     =$1000

SALES:  SELLER
NAME            =MARY SMITH
ASKING-PRICE    =$1000

SALES:  CLIENT
NAME            =JOHN SMITH
UPPER-LIMIT     =$1000

SALES:  SELLER
NAME            =GEORGE BROWN
ASKING-PRICE    =$1000
```

Data Item Types

Equated (joined) data items can be of different types and lengths, with the exception of character types, which may only be equated to other character types. If two items of different lengths are equated, QUERY treats the shorter one as if it were padded with blanks to match the length of the longer data item.

Null Connecting Data Items

The IMAGE subsystem initializes numeric type data items to zero, and character type data items to ASCII null. When data items are equated with a relational operator for retrieval, QUERY compares data item values based on their types. When comparing two numeric type data items, QUERY considers the two items equal when both items have a zero value. If the comparison is between character type data items with the values of ASCII null, QUERY does not consider the relationship equal, and no entries are joined.

Zoned Type Data Items

When joining zoned type data items, the zoned numbers with the sign overpunch are considered equal to the equivalent value without an overpunch. For example, 5A is considered equivalent to 51. When QUERY converts a different data type to zoned, it will pad with leading blanks. Leading nulls are not considered equivalent to leading blanks.

Packed Data Type Items

When QUERY converts a different data type to packed, it uses a COBOL Convention, a positive sign of 1100, a negative sign of 1101, and unsigned 1111. Positive and unsigned numbers are considered equivalent.

Data Set Equivalence

Equating data sets is a way of renaming a data set to enable the information contained in it to be used in more than one way. For example, assume you have the following simple data set EMP-DETAIL containing three data entries, and you want to create a REPORT that prints the employee's name and the employee's manager's name:

```
                          EMP-DETAIL
-----------------------------------------------------------------
   EMP-#                   EMP-NAME                   MGR-#
-----------------------------------------------------------------
   1792                     G. Smith                  1833
   1833                     H. Jones                  3421
   3421                J. President                   ****
-----------------------------------------------------------------
```

In this data set, the data item MGR# identifies both an employee and an employee's manager. For example, MGR# 1833 identifies G. Smith's manager (H. Jones) and is at the same time H. Jones employee number.

To create the desired report, the data set EMP-DETAIL must be equated to a dummy data set, and these two data sets connected with a JOIN command, to specify a compound data set from which the report can be generated.

The dummy data set MGR-DETAIL is a temporary name to which the existing data set is equated. Note that this equation must be specified after the data item equivalences.

```
>JOIN EMP-DETAIL.MGR-# TO MGR-DETAIL.EMP-#;&
>>   MGR-DETAIL = EMP-DETAIL
>MULTIFIND ALL
```

```
USING SERIAL READ
2  COMPOUND ENTRIES QUALIFIED
```

The data set equivalence above is not actually executed until a MULTIFIND or MULTIFIND ALL command is entered to retrieve some or all of the compound entries you have formed.

Two compound entries have been retrieved from which the desired report can be created.

```
>REPORT ALL

EMP:EMP-DETAIL
EMP-#             =1792
EMP-NAME          =G. SMITH
MGR-#             =1833

EMP:MGR-DETAIL
EMP-#             =1833
EMP-NAME          =H. JONES
MGR-#             =3421

EMP:EMP-DETAIL
EMP-#             =1833
EMP-NAME          =H. JONES
MGR-#             =3421

EMP:MGR-DETAIL
EMP-#             =3421
EMP-NAME          =J. PRESIDENT
MGR-#             =****
```

Below is another report showing employee and manager names:

```
>REPORT
>>H1,"Employee Name",30
>>H1,"Manager Name",55,SPACE A2
>>D1,EMP-DETAIL.EMP-NAME,27
>>D1,MGR-DETAIL.EMP-NAME,55
>>END


          Employee Name             Manager Name


            G. SMITH                    H. JONES
            H. JONES                  J. PRESIDENT
```

Using the @ Parameter

The optional @ parameter allows you to preserve all of the data item values in one of the data sets of a data item equivalence. The values in the data set which is next to the @ (on the same side of the TO as the @) will be preserved when data sets are joined even if a corresponding data item value for the equated data items does not exist in the second data set. For example, given the following data sets:

```
        SALES-DETAIL                    STOCK-DETAIL
-----------------------------     -------------------------------
  ACCT-#    STOCK-#    QUAN         STOCK-#    DESCR    ON-HAND
-----------------------------     -------------------------------
  666         90       350           110         NUT      970
  222         60        25            60        BOLT     1200
-----------------------------     -------------------------------
```

The next JOIN command produces the following compound data set:

```
>JOIN SALES-DETAIL.STOCK# TO STOCK-DETAIL.STOCK#

<--------(SALES DETAIL)--------> <---------(STOCK-DETAIL)--------->
-------------------------------------------------------------------------
  ACCT-#    STOCK-#     QUAN           STOCK-#     DESCR      ON-HAND
-------------------------------------------------------------------------

   222        60         25             60         BOLT        1200
-------------------------------------------------------------------------
```

If you want to keep all of the entries in the SALES-DETAIL data set, use the @ parameter. Placing the @ sign on the SALES-DETAIL side of the TO ensures that all entries for the SALES-DETAIL data set will be included in the resulting compound data set even if there is no corresponding value for STOCK# in the STOCK-DETAIL data set. For example, the JOIN below produces the following compound data set.

```
>JOIN SALES-DETAIL.STOCK# @ TO STOCK-DETAIL.STOCK#

 <-------(SALES DETAIL)------><---------(STOCK-DETAIL)-------->
 ---------------------------------------------------------------------
   ACCT-#    STOCK-#    QUAN          STOCK-#    DESCR    ON-HAND
 ---------------------------------------------------------------------
     666        90       350             *        ****        *
     222        60        25            60        BOLT      1200
 ---------------------------------------------------------------------
```

The compound entry STOCK# = 90 is included, but there is no STOCK# = 90 in the STOCK-DETAIL set. The corresponding missing entry from STOCK-DETAIL is represented with asterisks. Missing entries can be retrieved using the MULTIFIND or SUBSET command with the $MISSING parameter.

Placing the @ sign on the other side of the TO preserves all entries in the data set STOCK-DETAIL, and marks missing entries from the SALES-DETAIL with asterisks. The next JOIN command produces the following compound data set.

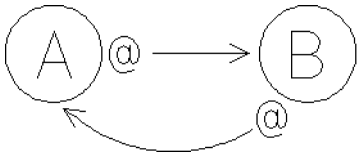
```
>JOIN SALES-DETAIL.STOCK# TO @ STOCK-DETAIL.STOCK#

 <---------(SALES-DETAIL)-------> <--------(STOCK-DETAIL)------>
 ---------------------------------------------------------------------
   ACCT-#    STOCK-#    QUAN          STOCK-#    DESCR    ON-HAND
 ---------------------------------------------------------------------
     *          *         *            110         NUT        970
   222         60        25             60        BOLT       1200
 ---------------------------------------------------------------------
```

Note: When forming compound entries for three or more data sets, once an entry from a data set is considered missing, no further joins can be made with the missing entry, and missing entries will be propagated.

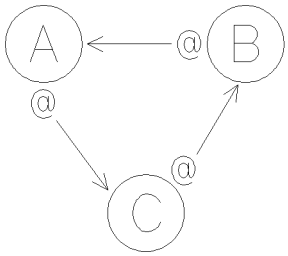
When joining data sets with the @ parameter, the following restrictions apply:

- The @ parameter is not allowed on both sides of the TO.
- The @ parameter is not allowed on a cycle of data sets. A cycle occurs when the data item equivalences, in effect, form a circle. The following are examples of illegal data cycles:

```
>JOIN SETA.ITEM1 @ TO SETB.ITEM1,&
>>     SETB.ITEM2 @ TO SETA.ITEM3
```




```
>JOIN SETA.ITEM1 TO @ SETB.ITEM2,&
>>     SETB.ITEM3 TO @ SETC.ITEM3,&
>>     SETC.ITEM2 TO @ SETA.ITEM2
```



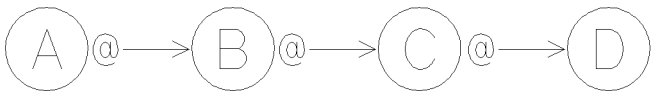
- When joining three or more data sets, the @ parameter must be propagated. Any data item equivalence naming a data set on the opposite side of the TO from the @ parameter must also have an @ sign associated with it, and the @ must be on the same side of the TO as that data set.

Propagating @ Signs

Example 1

In this example, an @ sign is associated with SETA in the equivalence between SETA and SETB. Therefore, the data set on the opposite side of the TO from the @ sign (SETB) must also have an @ sign associated with it. The @ sign associated with SETB requires that @ sign be associated with SETC. The @ sign associated with SETC requires that an @ sign be associated with SETD if SETD was joined to another data set. Since SETD is not connected to another data set, no @ sign is necessary.

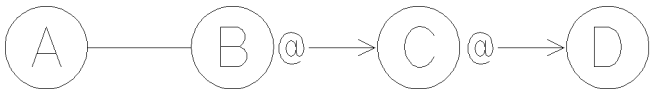
```
>JOIN SETA.ITEM1 @ TO SETB.ITEM2,&
>>     SETB.ITEM3 @ TO SETC.ITEM4,&
>>     SETC.ITEM5 @ TO SETD.ITEM6
```



Example 2

In the next example, the @ sign associated with SETB requires that an @ be associated with SETC. The @ associated with SETC requires that an @ be associated with SETD if SETD was joined to another data set. Since SETD is not connected to another data set, no @ sign is associated with it.

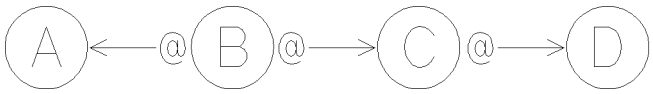
```
>JOIN SETA.ITEM1 TO SETB.ITEM2,&
>>     SETB.ITEM2 @ TO SETC.ITEM3,&
>>     SETC.ITEM4 @ TO SETD.ITEM5
```



Example 3

In this example, the @ signs associated with SETB require @ signs to be associated with both SETA and SETC when these data sets are connected with other data sets. Only SETC has an additional connection and requires an @ sign.

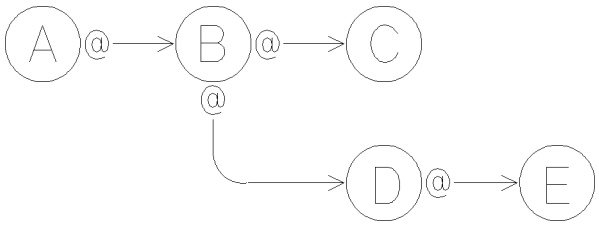
```
>JOIN SETA.ITEM1 TO @ SETB.ITEM2,&
>>     SETB.ITEM1 @ TO SETC.ITEM2,&
>>     SETC.ITEM1 @ TO SETD.ITEM4
```



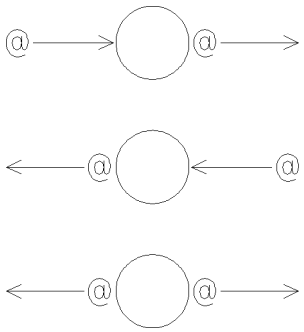
Example 4

In the equivalence between SETA and SETB, SETA has the @ sign associated with it. Therefore, every time SETB is connected to another data set, an @ must be associated with it. This accounts for the two @ signs associated with SETB. SETD must also have an @ sign because of its connection with SETB. SETC and SETE are not required to have associated @ signs because they have no further connections.

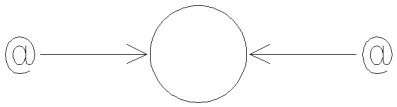
```
>JOIN SETA.ITEM1 @ TO SETB.ITEM2,&
>>     SETB.ITEM3 @ TO SETC.ITEM2,&
>>     SETB.ITEM4 @ TO SETD.ITEM1,&
>>     SETD.ITEM2 @ TO SETE.ITEM3
```



In summary, @ signs may form the following connections:



Note that @ signs may *not* take the following form:



Note It may be helpful to diagram the connections established by the JOIN command as shown in this section. When the @ parameter is used, a diagram can allow you to easily see where an @ must be propagated. A diagram will also reveal any illegal connections made with the @ parameter.

JOIN procedure

Executes a JOIN procedure stored in the current Proc-file.

Syntax

J[OIN]*procedure name* [,*character*]

For example:

```
>JOIN PROCA
```

Where*procedure name* = PROCA

```
>J USERS X
```

Where*procedure name* = USERS, *character* = X

Parameters

procedure name	is the name of a JOIN command previously stored as a procedure using the CREATE command. The procedure must exist in the current Proc-file specified in the PROC-FILE = command or in the response to the PROC-FILE prompt.
character	is any printable ASCII character. If *character* is included in the command, the JOIN procedure is listed.

Discussion

QUERY searches the current Proc-file (defined by a DEFINE or PROC-FILE command) and executes the procedure named in the command. If the Proc-file has not been declared, the procedure does not exist in the Proc-file, or the procedure is incorrect in some way, you are informed by an error message. If *character* is included in the command, QUERY prints the procedure on the standard list device before executing it.

For more information about storing and using JOIN procedures, refer to the CREATE command.

LANGUAGE=

Specifies the user language.

Syntax

$$\texttt{LA[NGUAGE]=}\left\{\begin{array}{l} langname \\ langnum \end{array}\right\}$$

For example:

```
>LANGUAGE=SPANISH

>LA=12
```

Parameters

langname	is the name of a supported user language.
langnum	is the user language ID number. Each language is associated with a unique ID number.

Discussion

The user language is different from the data base language. The user language is used for determining collating sequences and language dependent display characteristics. For example, the user language is used when sorting data in a report or when determining the thousands separator and decimal point when displaying numbers.

If no LANGUAGE= command is entered, the default language is NATIVE-3000. This means that QUERY will operate in the same way as before Native Language Support enhancements were made.

Each computer system supports a set of languages. Consult your system manager for a list of the configured languages on your system and the language ID numbers associated with them.

Refer to Appendix D for more information on Native Language Support.

LIST

Prints complete or partial entries from a single data set with automatic formatting and headings.

Syntax

L[IST][*data base name*:] { *data set name* / *data item list* } [FOR *relation* [{ AND / OR } *relation*] ...]

[END]

For example:

```
>LIST LABOR FOR BADGE# IE "09.18" AND F-NAME IE JOE
```

Where *data set name* = LABOR, *relation* = BADGE# IE "09.18", and *relation* = F-NAME IE JOE

```
>LIST BADGE#,F-NAME,L-NAME
```

Where *data item list* = BADGE#,F-NAME,L-NAME

```
>L LABOR
```

Where *data set name* = LABOR

Parameters

data base name is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of a data set in a currently open data base.

data item list is a list of either simple data item names or compound data items with an optional (*subscript*), separated by commas. All data items must be from the same data set, and can be listed in any order. A data item cannot be qualified with a data set name, but the first data item can be qualified with a data base name.

subscript is a number to indicate which sub-item you want to locate. *Subscript* is entered with parenthesis and must be an integer > 1 and <= to the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

relation takes the form:

[*data set name.*] *data item name* [(*subscript*)]

relop "value" [,*"value"*] . . .

data item name is the name of a data item contained in the data set. If you use *data set name* preceding the FOR, it is not necessary to qualify any data item names with the data set name. If you precede the FOR with a *data item list*, you can qualify the data item in the first relation with a data set name.

relop is a relational operator as shown in Table 3-3.

value is a data item value. It must be the same type and within the same value range as the data item named in the *relation*. A *value* need not be

enclosed in quotation marks (") unless the value contains special characters. Any *value* not contained within quotation marks is upshifted. For example, California is converted to CALIFORNIA before it is compared to data item values in the data base. *Value* must be an exact match for character type data items (type U and X). You may use null values. Refer to "Using Null Values" under the FIND command for more information.

`END` must be included in a procedure.

Table 3-3. LIST Command Relational Operators

OPERATOR	MEANING
`=` `IS` `IE` `EQ`	is equal to (Multiple *values* may be used with these operators.)
`#` `<>` `ISNOT` `INE` `NE`	is not equal to (Multiple *values* may be used with these operators.)
`<` `ILT` `LT`	is less than
`>=` `INLT` `GE`	is not less than (is greater than or equal to)
`>` `IGT` `GT`	is greater than
`<=` `INGT` `LE`	is not greater than (is less than or equal to)
`IB` $value_1$, $value_2$	is between (and including) $value_1$ and $value_2$
	Note: The operators <>, <=, and >= cannot have any intervening spaces (embedded blanks).

Discussion

The maximum number of logical connectors (AND, OR) which can be used in the LIST command is 10. Refer to "Logical Connectors" under the FIND command.

The LIST command prints all or a subset of the data item values from a single data set. It is one of the simplest ways to report on your data since you do not need to design a report or specify the format and headings.

LIST always uses a serial read.

Listing Format

The data is printed in columns. The width of the columns (or fields) is determined by the data item type. Table 3-4 summarizes field widths. QUERY provides two spaces between the fields.

Data item names are printed as column headings at the top of each page. If the complete data item name is longer than the field width, it is truncated. Headings of character type data items are left-justified and numeric types are right-justified.

If all of the data you request does not fit on one line (in one record), data items at the end of the *data item list* or the data entry are ignored. The line length varies with the device you are using. It is usually 72 to 80 characters for a terminal and a maximum of 136 characters for a line printer.

Table 3-4. Field Widths of Data Item Types

ITEM TYPE	FIELD SIZE (in characters)
I1	6
I2	11
I4	20
J1	5
J2	10
J4	19
K1	5
K2	10
R2	12
R4	22
Z*n*	*n*+1 (maximum = 20)
P*n*	*n* (maximum = 20)
U*n*	*n* (maximum = line length) *
X*n*	*n* (maximum = line length) *
	* Absolute maximum = 136

Listing a Subset of the Data

You can list a subset of the data in three ways:

1. Use the *data item list* form of the commands to specify particular items you want to list. For example:

   >LIST F-NAME,L-NAME,SERVICEYRS

   lists the values of F-NAME,L-NAME, and SERVICEYRS for each entry in the set. This form is useful if the complete data entry does not fit on one line. You can change the order of the data items and print the last ones in the data entry first or print only the last items.

   * Use the FOR parameter to set criteria for selection of entries from the set. For example:

   >LIST LABOR FOR SERVICEYRS GE 5

   lists the value of all data items in each entry of the LABOR data set containing SERVICEYRS values greater than or equal to 5.

   * It is also possible to combine these techniques. For example:

   >LIST F-NAME,L-NAME FOR SERVICEYRS GE 5

   lists the full names of each person entered in the set whose years of service (SERVICEYRS) total is greater than or equal to 5. Note that the data item used as selection criteria need not be in the data item list.

Determining the Data Set to be Used

If you use the LIST command form specifying a data set name, there is no ambiguity as to which information will be listed. If you use a *data item list*, you should consider the following:

- If the command has a FOR clause, QUERY uses the data item in the first relation to determine the data set to be listed.
- If there is no FOR clause, QUERY uses the last item in the *data item list*.

In either case, if the data item appears in only one data set, that set (or a subset of it) is listed. If it appears in more than one set and it is qualified, the named data set is listed. Otherwise, QUERY uses the data set list and follows the rule described with the DATA-SET= command.

The Relation of LIST and FIND

The entries selected by the LIST command are not available for any other purpose except the output of this command. The entries selected by the most recent FIND command are unaffected by LIST and are still available for use with UPDATE and REPORT commands.

Examples

Example 1

In the example below, QUERY could not locate the required entry until the value was entered with the correct spacing.

```
>L CUSTOMER FOR STREET-ADDRESS="  868       DOYLE ROAD"

>L CUSTOMER FOR STREET-ADDRESS="868 DOYLE ROAD"

 ACCOUNT     LAST-NAME     FIRST-NAME     IN     STREET ADDRESS

10034765     SLATER        GENEVA         K      868 DOYLE ROAD
```

Example 2

In the next example, the data item values for INVENTORY with STOCK# equal to 6650D22S are listed.

```
>L INVENTORY FOR STOCK#=6650D22S

STOCK#   ONHANDQTY    SUPPLIER        UNIT-COS    LASTSH  BIN

6650D22S      5306    ACME               1427    120385    3
6650D22S       600    HEWLETT-PACKARD   12500    111585    3
6650D22S         3    H & S SURPLUS         0    121585    0
6650D22S       999    H & S SURPLUS      1500    120585    0
6650D22S        13    H & S SURPLUS      1445    121485    3
```

Example 3

In this example, the data item values for SALES with STOCK# equal to 6650D22S are listed.

```
>L SALES FOR STOCK#=6650D22S
```

```
 ACCOUNT    STOCK#    QUANTI     PRICE    TAX   TOTAL  PURCH-

24536173    6650D22S       3       598     20       0  120885
24566356    6650D22S       1     12500    750       0  121585
```

Example 4

This example lists CUSTOMER entries with ACCOUNT greater than 55555555.

```
>LIST CUSTOMER FOR ACCOUNT GT 55555555

 ACCOUNT   LAST-NAME     FIRST-NAME    IN   STREET-ADDRESS

76623455   MCFALL        JEFFEREY      X   6650 MONTEREY RD
74001813   FIELD         HUBERT        J   4556 GEARY
87654321   JONES         JOHN          P   1 PINE AVE
```

Example 5

Since all the values of each entry in the previous example do not fit on one line, this example lists by the data item names in order to get the items at the end of each entry.

```
>LIST LAST-NAME,STREET-ADDRESS,CITY,STATE FOR ACCOUNT GT 55555555
ACCOUNT        IS A MEMBER OF THESE SETS:
CUSTOMER,SALES
WHICH SET DO YOU WISH TO USE?
>>CUSTOMER

LAST-NAME      STREET-ADDRESS          CITY          ST

MCFALL         6650 MONTEREY ROAD      CARMEL        CA
FIELD          4556 GEARY              CUPERTINO     CA
JONES          1 PINE AVE              CAMPBELL      CA
```

Example 6

This example lists all STOCK# in the data set INVENTORY.

```
>LIST STOCK#
STOCK#         IS A MEMBER OF THESE SETS
PRODUCT,SALES,INVENTORY
WHICH SET DO YOU WISH TO USE?
>>INVENTORY

  STOCK#

6650D22S
2457A11C
3586T14Y
```

Example 7

Since QUANTITY is in the SALES data set and is part of the first relation, SALES data set STOCK# values are printed. SALES is automatically added to the data set list.

```
>LIST STOCK# FOR QUANTITY IGT 2 AND ACCOUNT ILT 88888888

  STOCK#

6650D22S
3586T14Y
5405T14F
7892Z43Y
```

Example 8

SALES and INVENTORY are both in the data set list so QUERY must prompt you for the data set.

```
>LIST STOCK#
STOCK#          IS A MEMBER OF THESE SETS
SALES,INVENTORY
WHICH SET DO YOU WISH TO USE?
>>INVENTORY

6650D22S
2457A11C
3586T14Y
5405T14F
6650D22S
7892Z43R
```

LISTREDO

Displays one or more commands in the command history buffer.

Syntax

$$\texttt{LISTR}\left[\texttt{EDO}\right]\left[m\right]\left[\texttt{,}n\right]\left\{\begin{array}{l}\texttt{;ABS}\\ \texttt{;REL}\\ \texttt{;UNN}\end{array}\right\}$$

Parameters

m,*n* displays a range of commands in the command history buffer. *m* is the starting number (least recent) and *n* is the ending number (most recent). If *m* is equal to *n*, only the specified command is displayed. *m* and *n* can either be absolute (positive) numbers or relative (negative) numbers; however, both must be either relative or absolute. The use of these parameters is discussed further in the table below.

START	END	RESULT
(omitted)	(omitted)	All commands in history buffer are displayed.
m	, *n*	Command *m* through command *n* are displayed.
m	(omitted)	Command *m* through the last command are displayed.
(omitted)	, *n*	The first command through command *n* are displayed.

`ABS` displays the absolute command numbers. ABS is the default. Absolute numbers are positive numbers beginning with the first (least recent) command in the command history buffer. For example, 1 is the first command in the command history buffer.

`REL` displays the relative command numbers. Relative numbers are negative numbers beginning with the last (most recent) command in the command history buffer. For example, -1 is the last command in the command history buffer.

`UNN` suppresses command numbering.

Discussion

The LISTREDO commands displays the commands from least recent to most recent. Up to 20 commands can be displayed. The LISTREDO command is added to the command history buffer.

Example

In this example, the first LISTREDO command is issued when the command history buffer contains two commands. However, when the buffer is displayed, three commands are listed since the LISTREDO command is added to the buffer. The next LISTREDO command lists all the commands in the commands history buffer with relative numbers. Finally, the third LISTREDO command lists the third most recent command through the most recent command with relative numbers.

```
>LISTREDO
     1) DEFINE
     2) FIND FINDX
     3) LISTREDO
>LISTREDO ;REL
    -4) DEFINE
    -3) FIND FINDX
    -2) LISTREDO
    -1) LISTREDO ;REL
>LISTREDO -3 ;REL
    -3) LISTREDO
    -2) LISTREDO ;REL
    -1) LISTREDO -3 ;REL
>
```

MODE=

Changes the mode of access to the primary data base.

Syntax

M[ODE]= *mode number*

For example:

```
>MODE=3
```

Where*mode number* = 3

Parameter

mode number is an integer from 1 to 8 representing the access mode you want to use.

Discussion

You can use this command to change your mode of access to the primary data base. If the MODE= command is entered, QUERY first closes the current primary data base before attempting to open the data base with the new access mode.

Example

```
text
  >MODE=6''
  >AD INVENTORY''
  ILLEGAL ACCESS''
  >MODE=3''
  >AD INVENTORY''
  STOCK#          =>>6650D22S
  ONHANDQTY       =>>11
  SUPPLIER        =>>H & S SURPLUS
  UNIT-COST       =>>1395
  LASTSHIPDATE    =>>121585
  BINNUM          =>>3

  STOCK#          =>>//
  >
```

In the example above, mode 6 is specified. Since mode 6 does not allow write access to the primary data base, the user must change to a mode which allows such access. Then QUERY allows the addition of an INVENTORY entry.

MULTIDB

Opens additional data bases and modifies their environment specifications.

Syntax

MULTID[B][*data base name*]

Parameter

data base name is the name of an IMAGE data base.

Discussion

MULTIDB enables you to open two or more data bases for simultaneous access, and can also be used to modify the environment of those data bases already opened with a MULTIDB command.

MULTIDB can specify an additional data base only after a primary data base has been defined with the DEFINE or DATA-BASE= command. Up to 10 data bases can be opened at one time, including the data base opened with the DEFINE or DATA-BASE= command.

Assume you have defined data base FIRST with the DEFINE command and want to access data base SECOND. You would enter:

```
>MULTIDB
DATA-BASE = >>SECOND
PASSWORD = >>CLERK
MODE = >>6
DATA-SETS = >>RETURN
DATA-BASE = >>//
```

The DATA-BASE, PASSWORD, MODE, and DATA-SETS parameters have the same meaning as in the DEFINE command. To clear the data sets list for a specific data base, enter two asterisks (**) after the DATA-SETS prompt. MULTIDB will continue to prompt you for these specifications until a carriage return or two slashes (//) is entered at the DATA-BASE= prompt. Entering a carriage return has no effect on the current environment specifications.

If you are signed on as the creator and enter a semicolon in place of the data base password, you will be given read and write class access to all data items and data sets in the specified data base. This is true even if there are no passwords specified for the data base.

If you enter an invalid password, you will be assigned a user class of zero which allows you read and/or write access to some or all of the data sets and data items in the specified data base. When this happens, the following message is printed:

```
PASSWORD DOES NOT MATCH ANY DEFINED FOR SPECIFIED DATA BASE;
USER CLASS ZERO (0) WAS ASSIGNED
```

If there are no data sets and/or data items which you can access, the following message will be returned:

```
BAD PASSWORD
```

If you are not sure that you will have access to the data items that you need, you can change the password with the MULTIDB command.

You must qualify the first data item name in the command with the appropriate data base name. If you are accessing a data base:

- other than the one opened with a DEFINE or DATA-BASE= command.
- with a command other than JOIN, MULTIFIND, SUBSET on a compound select file, or REPORT on a compound data set select file.
- with a command for which a data base name can be specified.

Only the first item need be qualified. If the item name is not qualified, QUERY assumes the data base to be the one opened with the DEFINE or DATA-BASE= command. If the data base is in another group and/or account, then the group and/or account name must be specified. The JOIN, MULTIFIND, SUBSET, and REPORT commands will prompt for a data base name if there is any ambiguity. The DBLIST= command may be used to specify which data base(s) to use.

The MULTIDB command can be used to open more data bases or to modify the environment specifications of a data base previously opened with MULTIDB. For example, if you wanted to open a third data base (THIRD) and modify the MODE of data base SECOND, you would enter:

Example

```
>MULTIDB
DATA-BASE = SECOND
DATA-BASE = >>RETURN
PASSWORD = ********
PASSWORD = >>RETURN
MODE = 1
MODE = >>2
DATA-SETS = SALES,INVENTORY
DATA-SETS = >>RETURN

DATA-BASE = >>THIRD
PASSWORD = >>MGR
MODE = >>6
DATA-SETS = >>CUSTOMER

DATA-BASE = >>//
```

MULTIFIND

Retrieves compound entries from a compound data set specified by the most recent JOIN command.

Syntax

```
MU[LTIFIND][#LIMIT=i;] { relation                                  }
                       { item identifier M[ATCHING] "pattern"      }
[ { AND } { relation                                  } ]
[ { OR  } { item identifier M[ATCHING] "pattern"      } ] ... [END]
```

Parameters

i is an integer specifying the maximum number of qualifying entries you want to retrieve. *i* must be >= 0. If you specify a negative number, QUERY ignores your input. When the #LIMIT = parameter is specified, only the first *i* qualifying entries are placed in the select file. If fewer than *i* qualifying entries exist, then all entries that qualify are put in the select file.

item identifier takes the form:

```
[data base name:] [data set name      ]  data item name
                  [dummy data set name]
[(subscript)]
```

data base name is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of a data set named in the most recent JOIN command.

dummy data set name is a temporary data set name established in the most recent JOIN command.

data item name is the name of a data item belonging to one of the data sets named in the most recent JOIN command. For matching, the data item must be type X or U.

subscript is a number to indicate which sub-item you want to locate. *Subscript* is entered with parentheses, and must be an integer >= 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

`MATCHING` allows you to retrieve data based on the comparison of data items with a specified "*pattern*".

"pattern" must be enclosed in quotation marks. Refer to the FIND command for further specifications.

relation takes the form:

$$[\,data\ base\ name:\,]\begin{bmatrix} data\ set\ name \\ dummy\ data\ set\ name \end{bmatrix}\ data\ item\ name$$

$$[\,(subscript)\,]\,relop\ \left\{\begin{matrix} "value" \\ \texttt{\$MISSING} \end{matrix}\right\}\left[\,,\left\{\begin{matrix} "value" \\ \texttt{\$MISSING} \end{matrix}\right\}\right]\ \ldots$$

data item name can be any data type. Refer to previous definition.

relop is a relational operator as shown in Table 3-1.

value is the data item value. It must be the same type and within the same value range as the data item named in the *relation*. *Value* need not be enclosed in quotation marks unless the value contains special characters. A *value* not contained in quotation marks is upshifted. For example, California is converted to CALIFORNIA before it is compared to data item values in the data base. *Value* must be an exact match for character type data items (type U or X). You can use null *values*. Refer to "Using Null Values" under the FIND command.

`$MISSING` is used to retrieve a missing data item value from a compound data set. $MISSING is discussed in detail later under the MULTIFIND command.

`END` must be included in a procedure.

Discussion

The MULTIFIND command retrieves data entries from compound data sets according to the selection criteria you give. This command can only be used to retrieve data entries that belong to a compound data set resulting from a JOIN command.

When a MULTIFIND command is entered, whether valid or invalid, the internal select file is cleared.

The MULTIFIND command does not lock the data sets named in the JOIN command for the duration of the formation of the compound data set, but only locks while reading data set entries. If you want locking to span the formation of the compound data set, use the SETLOCKS and RELEASE commands.

Note that entering a REPORT ALL command will show you the order in which the data sets were joined. This may not be the same order that you entered them in the JOIN command, as QUERY optimizes the actual joining of the data sets.

Up to 50 logical connectors (AND, OR) can be used in a MULTIFIND command. Refer to "Logical Connectors" under the FIND command.

Example

Suppose you have three data sets, SALES-DETAIL, STOCK-DETAIL, and MANUF-DETAIL, and you want to produce a report that shows the quantity of sales of a product, the quantity on hand, and manufacturing information about this product. To access information from all three data sets, a compound data set must be defined with the JOIN command.

```
>JOIN SALES-DETAIL.STOCK# TO STOCK-DETAIL.STOCK#,&
>>    STOCK-DETAIL.STOCK# TO MANUF-DETAIL.STOCK#
```

The JOIN above combines the three data sets into one compound data set using the common data item STOCK#. Until a MULTIFIND or MULTIFIND ALL command is performed, the compound data set is only defined, but not created. If you could look at the compound data set that would result from the above JOIN command, it would appear as follows:

```
<---(SALES-DETAIL)--->   <---(STOCK-DETAIL)--->  <---MANUF-DETAIL)--->
-------------------------------------------------------------------------
ACCT#   STOCK#   QUAN   STOCK#   DESCR  ON-HAND   STOCK#   LABOR  MACHINE
-------------------------------------------------------------------------
 111      50     100      50      NAIL    1000      50        5      1
 111      60      20      60      BOLT    1200      60       10      5
 222      50       5      50      NAIL    1000      50        5      1
 222      60      25      60      BOLT    1200      60       10      5
 222      70      95      70    WASHER     325      70       15      6
 333      50      45      50      NAIL    1000      50        5      1
-------------------------------------------------------------------------
```

Performing the following MULTIFIND creates the compound data set from which the entry below is retrieved.

```
>MULTIFIND QUAN > 80 AND ON-HAND > 500


 ------------------------------------------------------------------------
 ACCT#   STOCK#   QUAN   STOCK#   DESCR  ON-HAND   STOCK#   LABOR  MACHINE
 ------------------------------------------------------------------------
  111      50     100      50      NAIL    1000      50        5      1
 ------------------------------------------------------------------------
```

Using the $MISSING Parameter

The MULTIFIND command can contain the optional $MISSING parameter, which allows you to retrieve a missing (non-compound) entry from a compound data set. Missing entries result when the JOIN command contains the optional @ parameter. For more information on missing entries, refer to the JOIN command.

A missing entry is retrieved from a compound data set by comparing a data item with the value $MISSING in a relation. This parameter allows the relation to be evaluated as true if, for a particular compound entry, the non-compound entry that the data item belongs to is missing. Note that a relation with $MISSING will be evaluated as false if, for a particular compound entry, the non-compound entry for the data set to which the data item belongs is not missing, but the value of the data item is null.

Example

The following JOIN command produces the compound data set below.

```
>JOIN SALES-DETAIL.STOCK# @ TO STOCK-DETAIL.STOCK#


<--------(SALES-DETAIL)--------> <--------(STOCK-DETAIL)--------->
-----------------------------------------------------------------
  ACCT#     STOCK#     QUAN          STOCK#     DESCR    ON-HAND
-----------------------------------------------------------------
  111         50       100             50        NAIL     1000
  111         60        20             60        BOLT     1200
  222         50         5             50        NAIL     1000
  222         60        25             60        BOLT     1200
  222         70        95             70      WASHER      325
  333         50        45             50        NAIL     1000
  444         80        92             **        ****     ****
```

```
-----------------------------------------------------------------
```

The following MULTIFIND command does not retrieve any entries from the join.

```
>MULTIFIND ACCT# = 444 AND ON-HAND >0
```

The MULTIFIND command with the $MISSING parameter retrieves the following entry.

```
>MULTIFIND ACCT# = 444 AND ON-HAND > 0 OR&
>>         ACCT# = 444 AND ON-HAND = $MISSING


-----------------------------------------------------------------
  ACCT#     STOCK#     QUAN          STOCK#     DESCR    ON-HAND
-----------------------------------------------------------------
   444        80        92            **       ****      ****
-----------------------------------------------------------------
```

Methods of Retrieval

QUERY retrieves data from compound data sets with either a Keyed, Serial, or Sort/Merge method, or a combination of these. QUERY notifies you with an appropriate message on your terminal screen. A Keyed retrieval occurs when master or detail data set search items can be used to locate entries. A Serial retrieval occurs when the formation and searching of the compound data set requires serial scanning without benefit of search items. A Sort/Merge retrieval occurs if the compound data set requires sorting and merging of two or more data sets.

Because the Sort/Merge method of searching and retrieval may take considerable time, it is generally advisable to avoid it when accessing multiple data sets. You can abort a search by entering a (CONTROL) Y. QUERY will print the number of qualifying entries and ask if you want to continue searching. A "NO" reply will store the retrieved entries in an internal select file and discontinue the search. Search time can be limited with the #LIMIT = parameter.

Sort/Merge retrieval will not be used if all data items named in the data item equivalences are search items, and no @ signs are used. It is advisable to maximize the number of data item equivalences with the above properties. Refer to the JOIN command for information on using the @ sign in joining data sets.

MULTIFIND ALL

Retrieves all compound entries from a compound data set specified by the most recent JOIN command.

Syntax

MU[LTIFIND]ALL [#LIMIT=*i*;]
[[*data base name:*][*data set name.* / *dummy data set name.*] *data item name*]

Parameters

i	is an integer specifying the maximum number of qualifying entries you want to retrieve. *i* must be >= 0. If you specify a negative number, QUERY ignores your input. When the #LIMIT = parameter is specified, only the first *i* qualifying entries are placed in the select file. If fewer than *i* qualifying entries exist, then all entries that qualify are put in the select file.
data base name	is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command, and named in the most recent JOIN command.
data set name	is the name of a data set used in the most recent JOIN command. If *data base name* is specified, the data set must belong to that data base.
dummy data set name	is a temporary data set name established in the most recent JOIN command.
data item name	is a data item belonging to one of the data sets named on the most recent JOIN command. If *data set name* is specified, the data item must belong to that data set.

Discussion

The MULTIFIND ALL command retrieves all data entries from a specified compound data set resulting from a JOIN command.

```
>JOIN SALES-DETAIL.STOCK# TO&
>>    INVENTORY-DETAIL.STOCK#
```

If you could look at the compound data set resulting from the above JOIN command, it would appear like this:

```
<-------(SALES-DETAIL)--------> <--------(INVENTORY-DETAIL)-------->
-----------------------------------------------------------------
  ACCT#    STOCK#    QUAN            STOCK#    DESCR    ON-HAND
-----------------------------------------------------------------
  111        50      100               50       NAIL     1000
  111        60       20               60       BOLT     1200
-----------------------------------------------------------------
```

Example

Performing a MULTIFIND ALL on the data set SALES creates the compound data set from which the following compound entries are retrieved:

```
>MULTIFIND ALL SALES-DETAIL.STOCK#
USING SERIAL READ
2  COMPOUND ENTRIES QUALIFIED

>REPORT ALL

TOOLS: SALES-DETAIL
ACCT#           =111
STOCK#          =50
QUAN            =100

TOOLS: INVENTORY-DETAIL
STOCK#          =50
DESCR           =NAIL
ON-HAND         =1000

TOOLS: SALES-DETAIL
ACCT#           =111
STOCK#          =60
QUAN            =20

TOOLS: INVENTORY-DETAIL
STOCK#          =60
DESCR           =BOLT
ON-HAND         =1200
```

Note that a REPORT ALL will show you the order that the data sets were joined together. This may not be in the order that you specified them in the JOIN command as QUERY optimizes the actual joining of the data sets.

MULTIFIND procedure

Executes a MULTIFIND procedure stored in the current Proc-file.

Syntax

MU[LTIFIND] *procedure name* [,*character*]

For example:

```
>MULTIFIND SALESREP
```

Where*procedure name* = SALESREP

```
>MU USERS,X
```

Where *procedure name* = USERS, and *character* = X

Parameters

procedure name	is the name of a MULTIFIND command previously stored as a procedure using the CREATE command. The procedure must exist in the current Proc-file defined by the PROC-FILE = or DEFINE command.
character	is any printable ASCII character. If character is included in the command, then the MULTIFIND procedure is printed on the standard list device before it is executed.

Discussion

QUERY searches the current Proc-file and executes the procedure named in the command. If the Proc-file has not been declared, the procedure does not exist in the Proc-file, or the procedure is incorrect in some way, you are informed by an error message. For more information on storing and using procedures, refer to the CREATE command.

If null data values appear in the procedure, QUERY prompts you for the necessary values. In job mode, you must anticipate the prompts and supply the necessary values on separate lines in the job file. Refer to "Using Null Values" under the FIND command for further discussion.

The MULTIFIND procedure uses either SERIAL READ or SORT/MERGE to retrieve data. QUERY will print a message to inform you of the method used. Refer to the MULTIFIND command for more information on methods of retrieval.

OUTPUT=

Selects the output device for the FORM, HELP, DISPLAY, LIST, REPORT, and VERSION commands.

Syntax

O[UTPUT]= { TERM / LP }

For example:

```
>OUTPUT=LP

>O=TERM
```

Options

TERM	indicates that output is to be displayed on the device specified as $STDLIST. ($STDLIST is normally a terminal in session mode and a line printer in job mode.)
LP	indicates that output is to be sent to the file designated as QSLIST. QSLIST is equated to the MPE device class LP (line printer) unless you use an MPE :FILE command to specify otherwise.

Discussion

The default output device for the commands listed above is $STDLIST unless you use this command to change it. You can also use this command to change the output device back to $STDLIST at any time after setting it to QSLIST.

If you want to change the device associated with QSLIST while operating QUERY in session mode, you can use the (BREAK) key or terminate QUERY to enter the MPE :FILE command, or you can enter the :FILE command before running QUERY.

If you are operating in job mode, you can place a :FILE command in your job preceding the :RUN QUERY command.

All error messages and QUERY prompts are always sent through the $STDLIST device, regardless of whether OUTPUT=TERM or OUTPUT=LP.

Using the MPE FILE Command

If you want to associate the QSLIST file with a device other than the line printer, you must use the MPE :FILE command and then set OUTPUT=LP. For example:

```
:FILE QSLIST;DEV=device type
```

Device type is a name referring to a class of devices or a specific device. The different device classes and names are set up by the system manager at the time the MPE operating system is configured. You must ask the system manager which name to use for the device you want to associate with QSLIST.

QUERY will not output a record longer than 136 bytes (characters) even though the maximum record length for the device may exceed this limit.

The following :FILE command is used to send the output to a magnetic tape. In this case, TAPE is the name of the class of devices known as magnetic tapes.

```
:FILE QSLIST;DEV=TAPE
```

To send output to a disc file, follow the instructions below.

- If QSLIST is not a permanent file, you can specify the following file command:

  ```
  :FILE QSLIST;DEV=DISC
  ```

  QUERY will build the new file as an ASCII file with variable length records of 136 bytes. The first byte contains carriage control for the file. If you want to override these defaults you must specify what you want in the file equation. For example, the following will create an ASCII file named QSLIST with carriage control and fixed length records of 80 bytes.

  ```
  :FILE QSLIST;DEV=DISC;REC=-80,,F,ASCII;CCTL
  ```

- If QSLIST is already a permanent file or you want to send the output to a file with a different name, you can specify the following :FILE command. *fname* is the name of the file you want to create:

  ```
  :FILE QSLIST=fname,NEW;DEV=DISC
  ```

  If you use the NEW option in a :FILE command and the file already exists, QUERY prints the following message:

  ```
  QSLIST ALREADY EXISTS, PURGE OLD FILE?
  ```

  If you respond with "Y", QUERY overwrites the existing file. If you respond with "N", a file system error message is returned and OUTPUT is reset to TERM.

  If you want QUERY to append to an existing file, you must use the APPEND option, as shown below.

  ```
  :FILE QSLIST=fname,OLD;DEV=DISC;ACC=APPEND
  ```

Refer to the *MPE Commands Reference Manual* for information on using the FILE command.

Examples

Example 1

In this example, QUERY is running in session mode. QSLIST is equated to a terminal initially. When the output device to QSLIST is changed to LP, the INVENTORY data set is listed on the line printer. When the output device is set to the terminal again, the INVENTORY data set is listed on the terminal.

```
>OUTPUT=LP
>LIST INVENTORY
>O=TERM
>LIST INVENTORY


STOCK#        ONHANDQTY    SUPPLIER         UNIT-COS    LASTSH   BIN

6650D22S           5306    ACME WIDGET          1427    120385    3
2457A11C       11001345    ACME WIDGET          5031    120185    1
3586T14Y            144    CARDINAL MILLS        249    112085    2
```

Example 2

The next example illustrates the method for sending output to a tape device. In this case, a procedure from the Proc-file is listed first on the terminal and then on a tape. The BREAK key is used to return to MPE. QSLIST is equated to a magnetic tape device. QUERY execution is resumed and the output device is changed to QSLIST. The procedure is written to the QSLIST device and the output device is set to the terminal again.

```
>DISPLAY FNAMES

PROCEDURE: FNAMES

001   FIND LAST-NAME="" END

>BREAK
:FILE QSLIST;DEV=TAPE
:RESUME
READ pending
RETURN
>OUTPUT=LP
>DISPLAY FNAMES
>OUTPUT=TERM
```

Example 3

In this example, one entry is retrieved with the FIND command, and is displayed with the REPORT ALL command. Next, the BREAK key is used to temporarily return to the operating system. The FILE command redirects QSLIST to a disc file named SAVER which QUERY creates. QUERY execution is resumed. Finally, the OUTPUT=LP and REPORT ALL commands are used to write data to the SAVER file.

```
>FIND CUSTOMER.LAST-NAME IS CORCORAN END
USING SERIAL READ
1  ENTRIES QUALIFIED
>REPORT ALL


ACCOUNT         =54283540
LAST-NAME       =CORCORAN
FIRST-NAME      =CLIFFORD
INITIAL         =C
STREET-ADDRESS  =6105 VALLEY GREEN DR.
CITY            =CARMEL
STATE           =CA
ZIP             =93921
CREDIT-RATING   = 7.10000

>BREAK
:FILE QSLIST=SAVER,NEW;DEV=DISC
:RESUME
READ pending
RETURN
>OUTPUT=LP
>REPORT ALL
>
```

PASSWORD=

Changes your password for the primary data base.

Syntax

PA[SSWORD]= [*password*]

Parameters

password is the word you have been given by the data base administrator to use when accessing the data base. You must enter this word exactly as it was created. A distinction is made between passwords entered in lowercase or entered in uppercase.

Discussion

You can change your password, thus changing the group of data to which you have access. QUERY first closes the current data base before attempting to open the data base with the new password.

If you are signed on as the creator and enter a semicolon in place of the data base password, you will be given read and write access to all data items and data sets in the specified data base. This is true even if there are no passwords specified for the data base.

To take advantage of the non-echoing feature of QUERY, enter the PASSWORD= command without the password parameter and press RETURN. QUERY will prompt you for the password and, when entered, it will not be echoed back.

If you enter an invalid password, you are assigned a user class zero that allows you read and/or write access to some or all of the data sets or data items in the specified data base. When this happens, the following message is returned.

```
PASSWORD DOES NOT MATCH ANY DEFINED FOR THE SPECIFIED
DATA BASE; USER CLASS ZERO (0) WAS ASSIGNED
```

If there are no data sets and/or data items which you can access, the following message is returned.

```
BAD PASSWORD
```

If you are not sure that you will have access to the data items you need, you can change the password with the DEFINE, DATA-BASE=, or PASSWORD= command.

```
>PA=CLERK
>AD INVENTORY
ILLEGAL ACCESS
>PA=DO-ALL
>AD INVENTORY
STOCK#          =>>
  .
  .
  .
```

In the example above, the CLERK password is specified. The request to add an entry to the INVENTORY data set is rejected since CLERK does not allow write access to it. When the password is changed to DO-ALL, QUERY allows the user to add to the INVENTORY data set.

PROC-FILE =

Specifies the name of the current Proc-file.

Syntax

P[ROC-FILE]= *filename* [,*n*]

For example:

```
>PROC-FILE =FILEP,50
```

Where *filename* = FILEP and *n* = 50

Parameters

filename is the name of an MPE ASCII file. The file may reside in any group or account, as long as you are allowed read and lock access to it through the MPE file security. If you do not have read and lock access a file system error 93, (FS-93) security violation, will occur. To specify a file that is not local to your group and account, you must use a fully-qualified file name in the form: *file.group.account*. For example, SPEC.PUB.SYS is a file named SPEC in the PUB group in the SYS account.

n is the number of records in the file. The file can be from 5 to 400 records in length depending upon the number and length of procedures to be stored. (The default value is 126.) Each procedure is stored on a record boundary, and occupies one or more records. If *n* is specified for an existing file, it is ignored.

Discussion

Procedures are stored in the Proc-file. Before using any of the QUERY procedure commands, you must specify the name of the Proc-file. This definition stays in effect until changed with the PROC-FILE = or DEFINE command or until execution terminates.

If the file name does not exist, QUERY issues a message and opens and saves a disc file of size *n* with the specified file name using a file code of 1070. Once a Proc-file has been declared, QUERY always uses that file when executing any of the commands listed in the following table. If you have not specified the Proc-file before entering one of these commands, QUERY prints an error message.

The following table shows the commands which are used to modify procedures and those which are used to execute procedures. All of these commands operate on the current Proc-file.

MODIFICATION	EXECUTION
ALTER	FIND procedure
CREATE	JOIN procedure
DESTROY	MULTIFIND procedure
DISPLAY	REPORT procedure
RENAME	SUBSET procedure
	UPDATE procedure

Examples

Example 1

In the following example DISPLAY LIST is used to list the procedures in the current Proc-file. MANPROC contains five procedures.

```
>PROC-FILE =MANPROC
>DISPLAY LIST
FIND1    FIND2    UPD1    REP4    REP5
```

Example 2

You can create a Proc-file with the PROC-FILE = command. If you do not specify the number of records QUERY creates a file with 126 records.

```
>PROC-FILE  =PROCX
FILE DOES NOT EXIST, BEING CREATED
```

REDO

Edits and executes a command in the command history buffer.

Syntax

RED [O] [*m* | *-n* | *string*]

Parameters

m — displays the command that has this absolute number. Absolute numbers are positive numbers beginning with the first (least recent) command in the command history buffer. For example, 1 is the first command in the buffer. (This is the number displayed by the LISTREDO command with the ABS option or with the default of absolute numbering.)

-n — displays the command that has this relative number. Relative numbers are negative numbers beginning with the last (most recent) command in the command history buffer. For example, -1 is the last command in the buffer. (This is the number displayed by the LISTREDO command with the REL option.) *-n* must be from -1 to -20.

string — displays the most recent command that begins with the specified string. The string can only contain alphanumeric characters. It cannot contain blanks or special characters such as `; , ( ) =` .

Discussion

When the REDO command is issued, you are placed into an "edit mode" to modify the specified command. The subcommands discussed next can then be used to change the displayed command. You type the subcommands on the line below the line you are currently editing. You can type the subcommands in either upper or lowercase characters. Any characters other than the subcommands are interpreted as replacement characters.

The REDO command is not recorded in the command history buffer. In addition, the REDO command cannot be used in a command file.

The command history buffer contains the last 20 commands issued during the current QUERY session. You can use the LISTREDO command to display the contents of the command history buffer with the number of each command.

Parameters

`B` or `S` — breaks the current line by moving the character above the B (or S) and all subsequent characters on the line to the next line. The second half of the line becomes the current line.

`D` — deletes the character above the D. You can delete multiple characters by typing a D below the first character to be deleted and another D below the last character to be deleted. You can also type D's consecutively under

each character to be deleted. The D subcommand can be followed by the I subcommand.

`H` lists all the valid editing subcommands, then redisplays the line you are currently editing.

`I` inserts one or more characters preceding the character above the I. Type I followed by the characters to be inserted.

`L` displays the complete command in its current state, then redisplays the line you are currently editing.

`R` replaces one or more characters beginning with the character above the R. Type R followed by the replacement characters.

`U` undoes any changes to the current command and executes the original command. Type U anywhere under the current line.

`X` executes the edited version of the command. You can type X at any time.

`+[`*n*`]` displays the *n*th line forward in the current command for editing. By default, *n* is 1. Therefore, + and +1 have the same effect.

`-[`*n*`]` displays the *n*th line backward in the current command for editing. By default, *n* is 1. Therefore, - and -1 have the same effect.

RETURN displays the next line for editing. If the current line is the last line, the edited command is executed.

The ampersand (&) continuation character is not used to continue a command that is being edited.

Examples

Example 1

In the example below, the LISTREDO command is first used to display the number of the command to be edited. The REDO command is issued with the number previously displayed by the LISTREDO command. Next, the subcommands are used to edit and execute the command.

```
>LISTREDO -3
    47) DEFINE
    48) LIST LAST-NAME, STREET-ADDRESS, CITY, STATE FOR CUSTOMER.LAST-NAME
        IB A,N AND STATE <> "NEW YORK" OR ACCOUNT GT 55555555
    49) LISTREDO
>REDO 48
LIST LAST-NAME, STREET-ADDRESS, CITY, STATE FOR CUSTOMER.LAST-NAME
                                     D     D
LIST LAST-NAME, STREET-ADDRESS, CITY FOR CUSTOMER.LAST-NAME
RETURN
IB A,N AND STATE <> "NEW YORK" OR ACCOUNT GT 55555555
                                              R66666666
IB A,N AND STATE <> "NEW YORK" OR ACCOUNT GT 66666666
X
                .
                .
                .
edited LIST command is executed
```

Example 2

In the next example, the string parameter is used to specify the command to be edited. First, five commands in the buffer are displayed with the LISTREDO command. When the REDO command is executed, the most recent command that begins with FIND is displayed.

```
>LISTREDO ,5
     1) DEFINE
     2) FIND ALL CUSTOMER.ACCOUNT
     3) MULTIDB
     4) FIND PROCF5
     5) LISTREDO ,5
>REDO FIND
FIND PROCF5
          I2
FIND PROCF25
(RETURN)
      .
      .
      .
edited FIND command is executed
```

RELEASE

Removes all locks set by SETLOCKS command.

Syntax

```
REL[EASE]
```

Discussion

The RELEASE command unlocks the data set(s) that were locked by the SETLOCKS command. If the command is entered while the SETLOCKS command is not in effect, you will receive the following message:

```
SETLOCKS COMMAND WAS NOT IN EFFECT
```

When data set locks are released, LOCKOPTION=OFF (overridden by the SETLOCKS command) will take effect with the same value it had prior to the SETLOCKS command.

RENAME

Changes the name of a procedure in the current Proc-file.

Syntax

REN[AME] *old procedure name, new procedure name*

For example:

```
>RENAME UPCUST,UPACCT
```

Where *old procedure name*=UPCUST and *new procedure name* =UPACCT

Parameters

old procedure name	is the name currently associated with the procedure.
new procedure name	is the name you want to use for the procedure in the future. Refer to the CREATE command for naming rules.

Discussion

Both procedure names must follow the rules defined in the CREATE command description. An error message is printed if *old procedure name* does not refer to an existing procedure in the current Proc-file.

```
>PROC-FILE  =PROCACCT
>DISPLAY LIST

FINDACCT       FINDSALE      UPCUST       REPACCT      REPSALE
>RENAME UPCUST,UPACCT
>DISPLAY LIST

FINDACCT       FINDSALE      UPACCT       REPACCT      REPSALE
```

REPORT

Lists data item values of entries located by a retrieval command in the format you specify.

Syntax

R[EPORT] *report statement*; ... END

For example:

```
 >REPORT
>>H1,"NAME LIST",20,SPACE A2
>>D,LAST-NAME,20
>>D,FIRST-NAME,30
>>END
```

or

```
>R H1,"NAME LIST",20,SPACE A2;D,LAST-NAME,20;D,FIRST-NAME,30;END
```

Where *header statement* = `H1,"NAME LIST",20,SPACE A2` and *detail statements* = `D,LAST-NAME,20` and `D,FIRST-NAME,30`

Parameter

report statements	consist of a sequence of header, detail, sort, group, total, register, edit, and output control statements, as outlined in Table 3-5. These statements are explained in detail later in this section. Statements can be entered on separate lines or on one line separated by semicolons.

Discussion

If you enter REPORT without following it with report statements, QUERY will prompt you for the statements until you enter an END at the prompt.

If the first report statement is in error, the REPORT command terminates.

The REPORT command is an extension of a retrieval command in that it prints a report of the data entries located by the last FIND, MULTIFIND, or SUBSET command.

REPORT output can be directed to any desired output device through the MPE :FILE command and the QUERY OUTPUT= command. Refer to the OUTPUT= command for further discussion.

You can specify your own user defined procedure(s) to enable your report to perform specialized tasks not provided by QUERY. This is an advanced capability used by programmers. Refer to Appendix F for further discussion.

Table 3-5. REPORT Statements

STATEMENT	FUNCTION
`Header`	Prints title, column headings, page numbers, time of day, and the date at the top of each report page.
`Detail`	Prints data item or register values or a character string in the column position specified.
`Sort`	Sorts data based on the value of a specified data item.
`Group`	Prints a data item value or character string whenever the value of an appropriate "sort item" changes.
`Total`	Prints count, average, or totals for logical groups or entire report.
`Edit`	Describes edit masks used to punctuate Group, Detail, or Total fields.
`Register`	Specifies an operation to be executed in Register n.
`Output control`	Specifies the report output parameters.

REPORT Statements

Statements can also contain additional parameters which perform such tasks as skipping to the top of the next report page, spacing between report lines, and indicating edit masks to be used to insert punctuation such as decimal points, dollar signs, etc., into values printed in the report. These options are described in Table 3-6.

Table 3-6. REPORT Statement Parameters

PARAMETER	FUNCTION	APPLICABLE STATEMENTS
print position	determines the rightmost print position (column number) for the *print element.* For character data, this is the rightmost character; for numeric data, it is the position of the least significant digit.	`Header, Detail, Group, Total`
`SPACE A[`*number*`]`	space *number* lines after printing the report line. If *number* is omitted, one line is spaced.	`Header, Detail, Group, Total`
`SPACE B[`*number*`]`	space *number* lines before printing the report line. If *number* is omitted, one line is spaced.	`Header, Detail, Group, Total`
number	is the number of lines to be spaced (from 1 to 5).	
`SKIP {A | B}`	skips to the top of the next report page after printing the report line (SKIP A) or before printing the report line (SKIP B).	`Detail, Group, Total`
`E{`*number*` | Z}`	indicates that either an edit mask defined in the identically numbered edit statement (E*number*) is used to punctuate a value or, if you use the letter Z, that leading zeros are to be suppressed. In the latter case, no edit statement is required.	`Detail, Group, Total`

Skipping and Spacing

When paging is in effect, the following rules govern skipping and spacing:

1. If a SKIP B and a SPACE B are both associated with the same output line, the SKIP B is processed before the SPACE B.
2. If there are not enough lines remaining on the current page to satisfy a SPACE B, a page is ejected and then the spacing is effected.
3. If a SKIP A and a SPACE A are both associated with the same output line, the SPACE A is ignored.
4. If there are not enough lines remaining on the current page to satisfy a SPACE A, it is treated as a SKIP A.
5. A SKIP A on the last line of the report will not be executed.
6. SPACE A and SPACE B are allowed on the same REPORT statement. The SPACE B is done first, then the SPACE A.

Designing a Report

Report formats vary according to their use. However, many reports assume the general format depicted in Figure 3-7. The TITLE and HEADERS describe the report and are printed at the top of each page along with the page number. HEADERS are usually used to describe the report columns.

The report body consists of DETAIL lines, GROUP TITLES, and TOTALS along with other descriptive labels. Normally, each detail line displays information from a single data entry, although information can appear on more than one line per entry. A DETAIL field can be edited to include commas, decimal points, dollar signs, and other punctuation characters.

DETAIL lines can be sorted and grouped according to the values of data items in the entry. For example, a sales report may list sales results by country, region, sales office, and finally by individual salesperson within each office. A GROUP TITLE can be printed whenever a "sort field" changes value. For example, when the country changes, the name of the country could be displayed as a GROUP TITLE. The title can be a series of characters or a data item value.

SUBTOTALS can be printed for logical groups (for example, for each sales office) and GRANDTOTALS for the entire report. These totals add, average, or count the DETAIL fields in each column of the report. Like DETAIL and GROUP fields, TOTAL fields can be edited with punctuation characters.

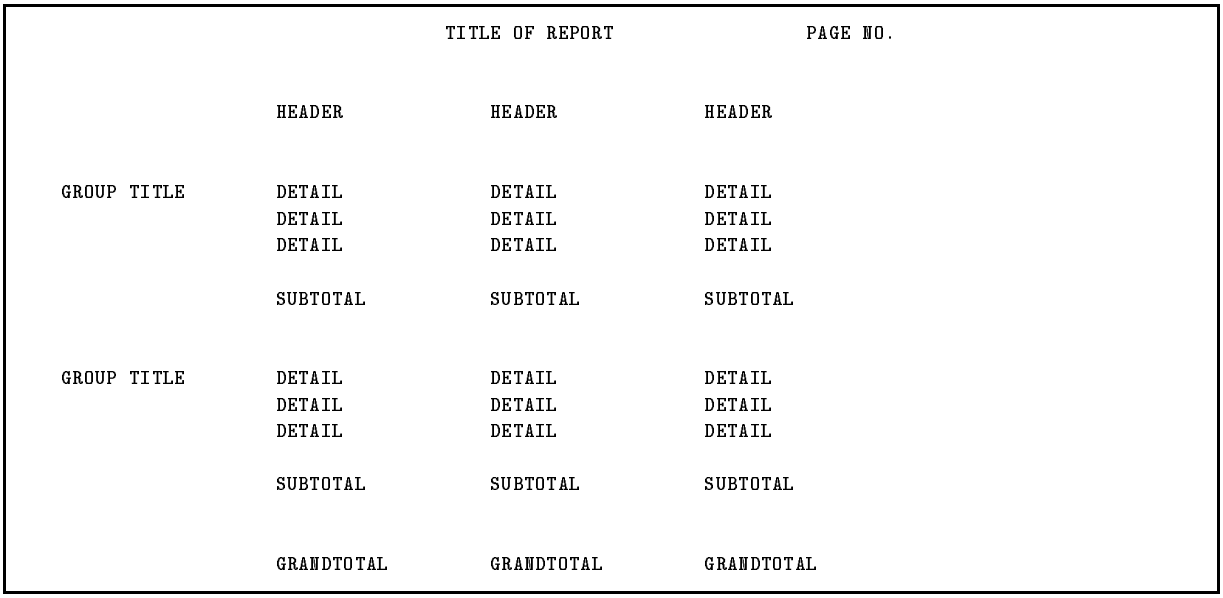


Figure 3-7. General Report Format

Through the following sections on REPORT Statements, a report is created by adding one statement type at a time and showing how the added statements change the report. Figure 3-8 contains the final version of the report.

```
AS OF: 01/07/86                                              PAGE 1
                        BOBO'S MERCANTILE
                        ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT

  0   H & S SURPLUS
                          7391Z22F     8/13/85           $5,012.50
                          5405T14F     9/11/85          $12,129.60
                          6650D22S    12/05/85          $14,985.00

     BIN TOTAL                                          $32,127.10  *


  1   ACME WIDGET
                          2457A11C    12/01/85     $553,477,666.95
      BAY PAPER CO.
                          7391Z22F    12/01/85           $4,704.00
      CARDINAL MILLS
                          5405T14F    11/28/85           $1,396.00



        JAKE'S JUNK
                          3739A14F    12/15/85           $1,189.32

       BIN TOTAL                                   $553,485,956.27 *


    2   ACME WIDGET
                          4397D13P     3/02/85          $55,080.00
        CARDINAL MILLS
                          3586T14Y    11/20/85             $358.56

       BIN TOTAL                                         $55,438.56 *


    3   ACME WIDGET
                          6650D22S    12/03/85          $75,716.62
        H & S SURPLUS
                          6650D22S    12/14/85             $187.85
                          6650D22S    12/15/85             $153.45

       BIN TOTAL                                         $75,057.92 *


       TOTAL INVENTORY                             $553,248,578.85  **
```

Figure 3-8. Sample Report

REPORT - HEADER STATEMENTS

Header statements are used to print report titles and column headings at the top of each report page.

Syntax

H*header number,print element,print position*[,SPACE A[*number*]] [,SPACE B[*number*]]
[,E{ *number* / Z }]

For example:

```
H2,F-NAME,20,SPACE A5
```

Where *header number* = 2, *print element* = F-NAME, *print position* = 20, and *number of spaces* = 5

Parameters

header number is an integer from 1 to 9. Up to nine lines of header information can be printed in addition to blank lines created by spacing before and after non-blank lines. Header information with the same header number is printed on the same line. The lowest-numbered header statement is printed first, the next-highest numbered statement is printed next. Header statements do not have to be consecutively numbered.

print element (1) is the name of a simple item or a compound item with an optional *subscript* parameter. The value of the specified data item is printed. The data item can be qualified with the data base and data set to which it belongs. The form is:

[*data base name:*] [*data set name.* / *dummy data set name.*] *data item name*
[(*subscript*)]

dummy data set name is a temporary data set name used in multiple data set access (refer to the JOIN command).

subscript is a number to indicate which sub-item you want to access. *subscript* is entered with parenthesis and must be an integer > 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

(2) is characters enclosed in quotation marks. The characters are stripped of the surrounding quotation marks and printed.

(3) is PAGENO, DATE, or TIME.

PAGENO	consecutively numbers each page of the report.
DATE	prints the date in the form: MMDDYY. The form of the date varies depending on the language.
TIME	prints the time in the form: HH:MM:SS.

Refer to Table 3-6 at the beginning of the REPORT command for descriptions of the other parameters.

Discussion

A header can contain up to 9 lines of information and any number of blank lines as long as it does not exceed the page size as defined by the output control statement LINES=. (Refer to the description at the beginning of this section.) A report consisting of only header statements will not generate any output.

Negative data item values of type P, Z, I, J, and K are output with a special character in the rightmost position, unless you use the NOPUNCH output control statement. This type of output is called overpunch. Refer to the REPORT ALL command for more information on overpunch.

```
>F ALL INVENTORY.STOCK#
USING SERIAL READ
13  ENTRIES QUALIFIED
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
        .
        .
        .
```

In the example above, entries are located first. Then report statements are entered describing the report headings. Characters in quotation marks are printed as they appear in the statement. DATE and PAGENO are generated and printed by QUERY. The following is an example of the specified report. The column and heading markers, in the example below, do not appear in an actual report.

```
           Col.    Col.        Col.        Col.   Col.         Cols.
            6       15          33          45     49          68 71
            |       |           |           |      |           |  |
            v       v           v           v      v           v  v
H1--> AS OF: 01/07/86                                       PAGE 1
H2-->                       BOBO'S MERCANTILE
H3-->                       ON HAND INVENTORY


H7-->BIN#  SUPPLIER         STOCK      SHIP DATE          INVENTORY
H8-->                                                        AMOUNT
```

REPORT - DETAIL STATEMENTS

Detail statements usually specify a data item name whose value changes with each data entry reported, although a fixed series of characters can be specified as well. The statement specifies the print position, top-of-form, line spacing, and applicable edit masks.

Syntax

D[*detail number*],*print element*,*print position*[,SPACE A[*number*]] [,SPACE B[*number*]]
[,SKIP { A / B }] [,E { *number* / Z }]

For example:

```
D2,BADGE#,35,SKIP B,E8
```

Where *detail number* = 2, *print element* = BADGE#, *print position* = 35, and *edit number* = 8

```
D,R3,15,SPACE A2
```

Where *print element* = R3, *print position* = 15, *number of spaces* = 2

Parameters

detail number is an integer from 1 to 99. If the number is omitted, the *print element* is printed on a group line when any control break occurs. (Refer to the discussion of both group and sort statements.) The lowest numbered statement is printed first and others follow in numeric sequence. Detail statements with the same number are printed on the same line. Information from a single data entry can therefore be printed on up to one hundred separate lines.

print element (1) is a simple data item name or a compound data item name with an optional *subscript* parameter. When a data item is specified, its value is printed for each entry reported. A data item can be qualified with the data base and data set to which it belongs. The form is:

[*data base name:*] [*data set name.* / *dummy data set name.*] *data item name*
[(*subscript*)]

dummy data set name is a temporary data set name used in multiple data set access. (Refer to the JOIN command).

subscript is a number that indicates which sub-item you want to access. *Subscript* must be an integer >= 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

(2) is the contents of a register (R*n*). If a register is specified, the data in the register is printed. You must enter both the letter and the *number*. Refer to REPORT Register Statement for more information.

(3) is a series of characters enclosed in quotation marks.

The characters are printed without the surrounding quotation marks, and are printed once for each entry reported.

Refer to Table 3-6 at the beginning of the REPORT command for descriptions of the other parameters.

Discussion

Negative data item values of type P, Z, I, J, and K are output with a special character in the rightmost position, unless you use the NOPUNCH output control statement. This type of output is called overpunch. Refer to the REPORT ALL command for more information on overpunch.

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48
>>END
```

If the same report shown in the header statement example has detail statements added, the result follows. The first detail statement prints the value of STOCK# ending in column 36. The next statement prints LASTSHIPDATE value on the same detail line ending in column 48. Note that detail lines are printed once per entry.

```
                                      Col.        Col.
                                      36           48
                                       |           |
                                       v           v
    AS OF: 01/07/86                                          PAGE 1
                              BOBO'S MERCANTILE
                              ON HAND INVENTORY


    BIN#  SUPPLIER             STOCK       SHIP DATE          INVENTORY
                                                                 AMOUNT
D1-->                          6650D22S      120385
D1-->                          2457A11C      120185
D1-->                          3586T14Y      112085
   .                           7391Z22F      120185
   .                           5405T14F      112885
   .                           7391Z22F       81385
   .                           5405T14F       91185
   .                           4397D13P       30285
   .                           3739A14F      121585
   .                           6650D22S      120585
   .                           6650D22S      121485
D1-->                          6650D22S      121585
```

REPORT - EDIT STATEMENTS

The edit statement is used for formatting data item values printed in a report. The statement performs such functions as suppressing leading zeros in numeric values, inserting characters such as dollar signs, dashes, commas, and decimal points. It also masks characters to eliminate them from the printed output. Edit statements can appear anywhere in the report.

Syntax

E*number,"edit mask"*

For example:

```
E3,"$$$999CR"
```

Where *number* = 3, and *edit mask* = "$$$999CR"

Parameters

number	is an integer from 0 to 9 identifying the edit statement. A report can have at most ten edit statements, each with a unique number.
"edit mask"	consists of from 1 to 20 characters (if the mask is to edit numeric data item values) or from 1 to the maximum length of the data item, not to exceed the maximum record length of the output device (if the mask is to edit alphanumeric data item values.) In either case, the characters must be surrounded by quotation marks. The length of the edit mask determines the length of the output field that is printed.

Alphanumeric Edit Masks

Alphanumeric edit masks consist of X's (used as place holders) and any other ASCII alphanumeric printing characters (used as insertion characters). QUERY examines the data item value specified in the detail, group, or total statement and the edit mask specified in the referenced edit statement, starting with the leftmost character of each.

If the character in the edit mask is X, a character from the data item value is printed in the corresponding position of the output field. If the character in the edit mask is any character other than an X, the edit mask character is printed in the corresponding position of the output field. For example, if the value ABCD is edited with the mask "X-X-X-X", the result is printed as A-B-C-D.

If there are fewer X's in the edit mask than there are in the data item value, the rightmost characters of the data item value that do not correspond to an X in the mask are omitted. For example, if the value ABCD is edited with the mask "XX", the result is printed as AB.

If there are more X's in the edit mask than there are characters in the data item value, QUERY prints asterisks in place of the unused X's in the edit mask. All insertion characters in the mask are printed in the output field. For example, if the value ABCD is edited with the mask "XXXX-X", the result is printed as ABCD-*.

Here are two more examples of alphanumeric edit masks:

DATA ITEM VALUE	EDIT MASK	PRINTED RESULT
A34B ABCD	"X//X-X-X-X" "- - - X"	A//3-4-B-* - - - A

Numeric Edit Masks

A numeric edit mask consists of the placement holders (9,Z,*,$), the sign characters (CR,-), and any other numeric ASCII printing characters used as insertion characters. Each of the place holders and sign characters serves a special purpose in editing data item values. Characters and their usage are specified in Table 3-7.

The numeric edit mask edits decimal integer values consisting of up to 20 characters (not counting the sign characters) in the combinations outlined in Table 3-8.

If the number of significant digits of the data item value is greater than the number of place holders (9,Z,*,$) in the edit mask, the output field is filled with asterisks. For example, if the value 12345 is edited with the mask "999CR" the result is *****.

Only one decimal point can appear in any edit mask.

If a minus sign appears in the edit mask in any position other than the rightmost character of the mask, the minus is treated as an insertion character. For example, if the value 12345 is edited with the mask 999-99, the result is 123-45.

Figure 3-9 shows the results of printing numeric data item values using numeric edit masks.

Table 3-7. Numeric Edit Mask Characters

CHARACTER	EXPLANATION
9	Each 9 in the edit mask is replaced with a decimal digit from the data item value in the corresponding position of the output field.
Z	Z is a zero suppression place holder. A Z in the edit mask is replaced with a decimal digit from the data item value in the corresponding position of the output field. If the data item value digit under consideration is a zero appearing to the left of the leftmost significant digit, QUERY inserts a blank in the output field, and all other zeros to the left of the significant digit are replaced by a blank in the output field.
*	* is an asterisk place holder. An * in the edit mask acts just like a Z with the exception that leading zeros in the data item value are replaced with asterisks in the output field.
$	$ is the dollar sign place holder. A $ in the edit mask acts just like a Z, except that the first zero in the data item value to the left of the leftmost significant digit is replaced with the dollar sign in the output field. All zeros to the left of the first leading zero are replaced with blanks in the output field.
CR	CR is a sign character, and always appears in the two rightmost positions of the edit mask. If the data item value is negative, QUERY prints the two characters (CR) in the first two rightmost positions of the output field. If the data item value is positive, QUERY prints two blank characters in place of the CR. No characters from the data item value are ever placed in the first two rightmost positions of the output field.
-	- is a sign character and acts the same as the CR characters. If the data item value is negative, QUERY prints the minus sign (-) in the rightmost position of the output field. If the data item value is positive, QUERY prints a blank in place of the minus. No character from the data item value is ever placed in the rightmost position of the output field.
Insertion characters	Insertion characters consist of any ASCII printing characters not previously mentioned. Insertion characters are printed in the output field in the same position they appear in the edit mask. Any insertion character appearing in the edit mask to the left of the leftmost significant digit of the data item value is replaced with blanks or an asterisk, depending upon which zero suppression character is specified in the edit mask. Only one decimal point can appear in an edit mask.

Table 3-8. Numeric Edit Mask Combinations

COMBINATIONS	EXAMPLE
9's only	"99" "999999" "9999"
Z's only	"ZZ" "ZZZZZZZZZZ"
Asterisks only	"****" "**********"
Dollar signs only	"$$$$" "$$$$$$$$$$$$$$"
Sign characters preceded by 9's	"9999CR" "999999-"
Sign characters preceded by 9's which are preceded by Z's, asterisks, or dollar signs	"ZZ99999-" "******9999999CR" "$$$999CR"
9's preceded by Z's, asterisks, or dollar signs	"ZZZZ9999" "***99999999" "$$$999999"
Any of the above combinations including insertion characters such as commas, one decimal point, slashes, etc. located anywhere in the edit mask, except to the right of sign characters.	"$$999,999.99-" "999-9999" "99/99/99" "$$9999.99CR"

```
DATA ITEM VALUE        EDIT MASK              PRINTED RESULT

   0059              "$$$,999"          $059
   001024            "ZZZ,ZZZ"                1,024
   -0010555          "$$,$$$.99CR"            $105.55CR
   00010555          "$$,$$$.99CR"            $105.55
   -0010555          "$$,$$$.99-"             $105.55-
   15039250          "$,$$$,$$$.99CR"         $150,392.50
   00049             "*****"                  ***49
   044240474         "999-99-9999"                  044-24-0474
   -2145             "$,$$$.99"         $21.45
```

Figure 3-9. Sample Output Using Numeric Edit Masks

Real Numbers

Real values can be printed in either fixed-point (xx.xxx) or floating-point (xx.xxxExx) form. The default is that R2 values less than .1 or greater than 10 to the sixth (10^6) and R4 values less than .1 or greater than 10 to the sixteenth (10^{16}) are printed in floating point (scientific notation). All other R2 and R4 values are printed in fixed-point notation. To override the default and specify that fixed point form always be used for real number values, you can use edit masks.

These edit masks work the same way as for other data types except that the placement of the decimal point determines where the number will be placed in the field. Note that fixed R2 values occupy up to eight characters and floating-point R2 values occupy up to 12 characters. R4 values occupy up to 18 and 22 characters for fixed and floating-point respectively.

REAL VALUE	TYPE	DEFAULT REPORT OUTPUT
0	R2	0.00000
8399607	R2	.839961E+07
123456	R2	123456.
.0034567	R2	.345670E-02

REAL VALUE	TYPE	EDIT MASK	REPORT OUTPUT WITH EDIT MASK
0	R2	"ZZZZZ9"	0
8399607	R2	"ZZZZZZ"	839610*
123456	R2	"999999"	123456
.0034567	R2	".9999999"	.0034567

* Rounding difference: For this number to come out as 8399607 it must be R4 precision.

Example

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48,E2
>><user|E2,"XX/XX/XX"|
>>END''
```

In the example above, LASTSHIPDATE is altered with edit mask E2. Edit mask E2 is defined as "XX/XX/XX". Now the report looks like this:

```
AS OF: 01/07/86                                              PAGE 1
                          BOBO'S MERCANTILE
                          ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT
                          6650D22S    12/03/85
                          2457A11C    12/01/85
                          3586T14Y    11/20/85
                          7391Z22F    12/01/85
                          5405T14F    11/28/85
                          7391Z22F     8/13/85
```

```
5405T14F     9/11/85
4397D13P     3/02/85
3739A14F    12/15/85
6650D22S    12/05/85
6650D22S    12/14/85
6650D22S    12/15/85
```

REPORT - SORT STATEMENTS

The sort statement specifies data items whose values are used to sort data entries when they are printed in the report. It also defines control break levels for use by group and total statements in the report.

Syntax

$$\texttt{S}\left[\,\textit{level}\,\right],\textit{data item name}\left[\,,\left\{\begin{array}{l}\texttt{ASC}\\\texttt{DES}\end{array}\right\}\right]$$

For example:

```
S3,BADGE#,ASC
```

Where *level* = 3 and *data item name* = BADGE#

```
S,L-NAME
```

Where *data item name* = L-NAME

Parameters

level — is an integer from 1 to 10. A sort statement with *level* greater than 1 must be accompanied by sort statements containing all lower level numbers. That is, if S3 appears in the report, S2 and S1 must also appear in it. You need not specify a level. In this case, the sort statement sorts but does not define a control break.

data item name — is the name of a simple data item or a compound data item with an optional *subscript* parameter, that is contained in data entries selected by the last retrieval command. If you used a FIND CHAIN to retrieve entries, use a *master* data item in the sort statement. It can also be qualified with the data base and data set to which it belongs. The form is:

$$\left[\,\textit{data base name:}\,\right]\left[\begin{array}{l}\textit{data set name.}\\\textit{dummy data set name}\end{array}\right]\ \textit{data item name}$$
$$\left[\,(\textit{subscript})\,\right]$$

dummy data set name is a temporary data set name used in multiple data set access. (Refer to the JOIN command).

subscript is a number that indicates which sub-item you want to access. *Subscript* is entered in parenthesis and must be an integer >= 1, and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

ASC — indicates that the data item values are to be ordered in ascending order. If you do not specify ASC or DES, the default order is ASC.

DES indicates that the data item values are to be ordered in descending order.

The following sort statement sorts the entries into the order specified by the value of L-NAME.

```
 S1,L-NAME

DATA ENTRIES AFTER FIND

     L-NAME    F-NAME     AGE

      WHITE     ROB       26
      BROWN     JACK      32
      GREEN     ROB       49
      WHITE     LARRY     81
      BROWN     CHRIS     17
      GREEN     SAM       28
      GREEN     BILL      45
      BROWN     DAN       39
      WHITE     WILL      22

DATA ENTRIES AFTER SORT EXECUTED
     L-NAME    F-NAME    AGE

      BROWN     JACK      32
      BROWN     CHRIS     17
      BROWN     DAN       39
      GREEN     ROB       49
      GREEN     SAM       28
      GREEN     BILL      45
      WHITE     ROB       26
      WHITE     LARRY     81
      WHITE     WILL      22
```

The higher-numbered sort statement identifies the major (or first) sort field, while the lower-numbered sort statement identifies the minor sort field. The minor sort arranges entries in the order specified, keeping all major sort items with identical values together, in other words, it sorts within subsets of the entire set of entries.

```
S1,F-NAME
S2,L-NAME
```

If the statements above appear in a report, the result would be as follows.

```
                        L-NAME                          F-NAME   AGE

level 2 (major)-->  BROWN       level 1 (minor)-->  CHRIS    17
Control break        BROWN       control breaks      DAN      39
                       BROWN                                  JACK     32
level 2 (major)-->  GREEN                                  BILL     45
control break        GREEN                                 ROB      49
                       GREEN                                  SAM      28
level 2 (major)-->  WHITE                                  LARRY    81
```

```
control break       WHITE                                ROB      26
                      WHITE                                WILL     22
```

Control Breaks

A control break occurs during the printing of a report whenever the value of a current entry for a data item defined in a numbered sort statement is different from the value of the last entry. When the first entry is printed, a control break occurs since the data item value changes from null (no value) to the first value. Totals are not printed when the first control break occurs.

In the previous examples, a control break occurs when the value of L-NAME becomes BROWN, when it changes to GREEN, and again when it changes to WHITE. This is known as a level 2 control break because the data item named L-NAME appears in a sort statement labeled S2. The level 1 control break is associated with the data item named F-NAME and sort statement labeled S1.

A control break occurs for all lower levels whenever a higher level control break occurs. For example, when a control break occurs for level 2 (L-NAME), a control break also occurs for level 1 (F-NAME).

A group or total statement prints only when a control break occurs that is at the same level as the group or total statement. This means that a total statement labeled T1 prints only when a level 1 control break occurs, or a group statement labeled G2 prints only when a level 2 control break occurs. Consult the descriptions of group and total statements later in this section for an explanation of their functions.

Sort statements with no *level* (i.e., no number) are used to sort entries but do not define control breaks for use by group or total statements.

Major to Minor Sort Fields

Numbered and unnumbered sort statements can appear in the same REPORT command. The order in which unnumbered sort statements appear in the *report body* is significant. The first unnumbered statement defines the most minor sort field, while the last unnumbered statement defines the most major sort field. QUERY defines sort fields in the following order from most major to most minor:

```
MOST MAJOR ----------------------------------------------------> MOST MINOR

S10 ---> S1 ---> S(last in report body) ---> S(first in report body)

    STATEMENTS                ORDER

    S2,OFFICE        MONTH    (S3)              MAJOR
    S,PARTNO         OFFICE   (S2)                |
    S1,SLSMAN        SLSMAN   (S1)                |
    S,QUANTITY       QUANTITY (S last)             v
    S3,MONTH         PARTNO   (S first)         MINOR
```

Maximum Number of Sort Items

The number of data items you can use to sort is limited in two ways. The maximum number of sort statements allowed in a single report is 66. You can have up to 66 sort statements provided that the combined length of the data items in all of the 66 statements is not greater than 2045 words.

Example

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48,E2
>>E2,"XX/XX/XX"
>>S2,BINNUM
>>S1,SUPPLIER
>>S,LASTSHIPDATE
>>END
```

In the example above, BINNUM is defined as a sort level 2, SUPPLIER as sort level 1, and LASTSHIPDATE as a sort without a control break. As shown in the new report, the detail entries are now sorted by BINNUM, SUPPLIER, and LASTSHIPDATE. The values for BINNUM and SUPPLIER will be printed in group statements which are described next.

```
AS OF: 01/07/86                                              PAGE 1
                          BOBO'S MERCANTILE
                          ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT
                          7391Z22F     8/13/85
                          5405T14F     9/11/85
                          6650D22S    12/05/85
                          2457A11C    12/01/85
                          7391Z22F    12/01/85
                          5405T14F    11/28/85
                          3739A14F    12/15/85
                          4397D13P     3/02/85
                          3586T14Y    11/20/85
                          6650D22S    12/03/85
                          6650D22S    12/14/85
                          6650D22S    12/15/85
```

REPORT - GROUP STATEMENTS

A group statement prints the value of a data item, the value in a register (R*n*), or a series of characters whenever a control break occurs.

Syntax

G*level,print element,print position* [,SPACE A[*number*]] [,SPACE B[*number*]]
[,SKIP { A }] [,E { *number* / Z }]

For example:

```
G3,WEEK,35,SKIP B,E2
```

Where *level* = 3, *print element* = WEEK, *print position* = 35, and *edit number* = 2

```
G1,R3,55,SPACE B2
```

Where *level* = 1, *print element* = R3, *print position* = 55, and *number of spaces* = 2

Parameters

level is an integer from 1 to 10 corresponding to the *level* of a sort statement.

print element (1) is the name of a simple data item or a compound data item with an optional (*subscript*) parameter. Data items can be qualified with the data base and data set to which they belong. The form is:

[*data base name:*] [*data set name* / *dummy data set name*] *data item name*
[(*subscript*)]

dummy data set name is a temporary data set name used in multiple data set access. (Refer to JOIN command.)

subscript is a number indicating which sub-item to access. *Subscript* is entered in parenthesis and must be an integer > 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

(2) is the contents of a register (R*n*).

(3) is a series of characters enclosed in quotation marks. You must specify the quotation marks, but QUERY will strip them when it prints the characters.

Refer to Table 3-6 at the beginning of the REPORT command for descriptions of the other parameters.

Discussion

Each control break occurs as a result of a sort statement labeled from 1 to 10. When the control break occurs, the group statement with the same number as the sort statement prints the information you specify. (Refer to "Control Breaks" under REPORT Sort Statements for more information.) Whenever a control break occurs, all group statements with a number equal to or less than the level of the sort statement causing the break will print a value and/or series of characters. All group statements print on the same line. Since a control break always occurs at the very beginning of the report, all group statements print their contents before any detail statements are executed.

If the REPORT command contains group statements but no sort statements, an error message is printed.

Negative data item values of type P, Z, I, J, and K are output with a special character in the rightmost position, unless you use the NOPUNCH output control statement. This type of output is called overpunch. Refer to the REPORT ALL command for more information on overpunch.

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48
>>E2,"XX/XX/XX"
>>S2,BINNUM
>>S1,SUPPLIER
>>S,LASTSHIPDATE
>>G2,BINNUM,3,SPACE B
>>G1,SUPPLIER,20
>>END
```

In this example, BINNUM will be printed when a control break occurs for sort level 2, and SUPPLIER when a sort level 1 control break occurs.

In the report below, notice that the two items mentioned in the detail statements do not print on the same line as the group statements. This is because the detail statements are numbered. Unnumbered detail statements print on the same line as a group statement whenever a control break occurs.

```
AS OF: 01/07/86                                              PAGE 1
                          BOBO'S MERCANTILE
                          ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT

  0   H & S SURPLUS
                          7391Z22F     8/13/85
                          5405T14F     9/11/85
```

```
                          6650D22S    12/05/85

1   ACME WIDGET       <-------- level 1 and level 2 control break
                          2457A11C    12/01/85
    BAY PAPER CO.          <--------- level 1 control break
                          7391Z22F    12/01/85
    CARDINAL MILLS         <--------- level 1 control break
                          5405T14F    11/28/85
    JAKE'S JUNK            <--------- level 1 control break
                          3739A14F    12/15/85

2   ACME WIDGET       <------- level 1 and level 2 control break
                          4397D13P     3/02/85
    CARDINAL MILLS
                          3586T14Y    11/20/85

3   ACME WIDGET
                          6650D22S    12/03/85
    H & S SURPLUS
                          6650D22S    12/14/85
                          6650D22S    12/15/85
```

REPORT - TOTAL STATEMENTS

The total statement prints a data item value, the value in a register (R*n*), a series of characters, or the total, average, or count of a group of data items whenever a control break occurs.

Syntax

T*level,print element,print position* [,SPACE A[*number*]] [,SPACE B[*number*]]

[,SKIP{ A / B }] [,E{ *number* / Z }] [{ ,ADD / ,AVERAGE / ,COUNT } [,NOREPEAT]]

Or the special form:

T*level,* R*n*

For example:

```
T4,WAGES,60,SKIP A,E2,ADD
```

Where *level* = 4, *print element* = WAGES, *print position* = 60, and *edit number* = 2

```
T1,"TOTAL WAGES=",40,SPACE B5
```

Where *level* = 1, *print element* = "TOTAL WAGES=", *print position* = 40, and *number of spaces* = 5

```
T3,R2
```

Where *level* = 3 and *n* = 2

Parameters

level is the letter F or an integer from 1 to 10 corresponding to the *level* of a sort statement. The letter F indicates the information is to be printed only at the end of the report after the last detail line and it relates to the entire report, not just a subgroup.

print element (1) is the name of a simple data item or a compound data item with an optional *subscript* parameter. Data items can be qualified with the data base and data set which they belong. The form is:

$$[\,\textit{data base name:}\,]\left[\begin{array}{l}\textit{data set name.}\\ \textit{dummy data set name.}\end{array}\right]\ \textit{data item name}\ [\,(\textit{subscript})\,]$$

dummy data set name is a temporary data set name used in multiple data set access (Refer to the JOIN command).

subscript is a number indicating which sub-item to access. *Subscript* is entered in parenthesis and must be an integer >= 1, and <= the number of sub-items defined for the compound item. QUERY will default to the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

(2) is the contents of a register (R*n*).

(3) is a number enclosed in quotation marks (a numeric literal). You must specify the quotation marks, but QUERY will strip them when it prints the characters.

`ADD` prints the control group total of the values for the data item specified as *print element*. The *print element* must be the name of a numeric type data item.

`AVERAGE` prints the control group average value for the data item specified as the *print element*. The *print element* must be the name of a numeric type data item.

`COUNT` prints a count of the number of values for that control group. The *print element* must be the name of a data item.

`NOREPEAT` can only be used if the REPORT command follows a MULTIFIND command. The NOREPEAT option allows the total statement to be executed for a compound entry only if the simple entry has not been previously encountered in the statement. By default, the total statement is executed for all compound entries. The number of registers which use the NOREPEAT option, plus the number of sort levels for which the TOTAL statement uses NOREPEAT, must be less than or equal to 10. Refer to the JOIN command for information about compound data sets. A more detailed explanation of this option appears later in this section.

`R`*n* is the register to be cleared.

Refer to Table 3-6 at the beginning of the REPORT command for descriptions of the other parameters.

Discussion

If the total statement is labeled TF, the ADD, AVERAGE, and COUNT options apply to all occurrences of the data item in the report.

Negative data item values of type `P`, `Z`, `I`, `J`, and `K` are output with a special character in the rightmost position, unless you use the NOPUNCH output control statement. This type of output is called overpunch. Refer to the REPORT ALL command for more information on overpunch.

If you use the special form of the command, specifying only a register without a *print element*, the register is cleared (reset to zero) when a control break occurs.

A control break results from a sort statement labeled from 1 to 10. When a control break occurs, the total statement corresponding to the sort level causing the break prints the information you specify. Total statements *not* labeled TF require corresponding sort statements. (Refer to the description of the sort statement for more information on control breaks.)

To perform more than one operation (total, average,count) on the same data item, you must specify a total statement for each operation. The information is printed on the same line if you use the same *level* for each statement. No more than five (5) data items can be used as print elements in total statements.

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48
>>E2,"XX/XX/XX"
>>S2,BINNUM
>>S1,SUPPLIER
>>S,LASTSHIPDATE
>>G2,BINNUM,3,SPACE B
>>G1,SUPPLIER,20
>>T2," *",70
>>T2,"BIN TOTAL",14,SPACE B,SPACE A
>>TF,"TOTAL INVENTORY",20,SPACE B3
>>TF," **",71
>>END
```

The total statements which have been added merely print character literals. The totals are computed with register statements which are described next. More total statements are added to the example in REPORT Register Statements.

In this example, the first total statement prints an asterisk in column 70 and the next one prints a character literal when a level 2 control break occurs. When the final totals are printed, the last two total statements print the specified characters. The resulting report appears as follows.

```
AS OF: 01/07/86                                          PAGE 1
```

```
                          BOBO'S MERCANTILE
                          ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT

  0   H & S SURPLUS
                          7391Z22F     8/13/85
                          5405T14F     9/11/85
                          6650D22S    12/05/85

      BIN TOTAL         <------ level 2 control break  *


  1   ACME WIDGET
                          2457A11C    12/01/85
      BAY PAPER CO.
                          7391Z22F    12/01/85
      CARDINAL MILLS
                          5405T14F    11/28/85
      JAKE'S JUNK
                          3739A14F    12/15/85

      BIN TOTAL                                          *


  2   ACME WIDGET
                          4397D13P     3/02/85
      CARDINAL MILLS
                          3586T14Y    11/20/85

      BIN TOTAL                                          *


  3   ACME WIDGET
                          6650D22S    12/03/85
      H & S SURPLUS
                          6650D22S    12/14/85
                          6650D22S    12/15/85

      BIN TOTAL                                          *



      TOTAL INVENTORY                                    **
```

The NOREPEAT Option

The NOREPEAT option is only applicable on compound data sets created by a MULTIFIND command. In a compound data set, a particular value can occur more than once. This is because each value of the data item on the left side of the TO (of the JOIN command) is being paired with each occurrence of that value of the data item on the right side of the TO.

If the JOIN and MULTIFIND commands are as follows, each STOCK# value in STOCK-DETAIL will be paired with each occurrence of that same STOCK# value in SALES-DETAIL.

```
>JOIN STOCK-DETAIL.STOCK# TO SALES-DETAIL.STOCK#
```

```
>MU ALL
```

For example, given the following values in the data sets STOCK-DETAIL and SALES-DETAIL:

STOCK-DETAIL

STOCK#	DESCR	ON-HAND
110	NUT	970
60	BOLT	1200
50	NAIL	1000
50	NAIL	900

SALES-DETAIL

ACCT#	STOCK#	QUAN
666	90	350
222	60	25
999	60	500
555	60	75
333	50	45
111	50	100

the following compound data set entries are created:

<------ (STOCK-DETAIL) --------><------- (SALES-DETAIL) -------->

STOCK#	DESCR	ON-HAND	ACCT#	STOCK#	QUAN
60	BOLT	1200	222	60	25
60	BOLT	1200	999	60	500
60	BOLT	1200	555	60	75
50	NAIL	1000	333	50	45
50	NAIL	1000	111	50	100
50	NAIL	900	333	50	45
50	NAIL	900	111	50	100

The one occurrence of STOCK# 60 in STOCK-DETAIL is paired with each occurrence of STOCK# 60 in SALES-DETAIL resulting in 3 entries in the compound data set. You can see by the ACCT# that each of these entries is separate and unique.

Now suppose you wanted to produce a report from this retrieval in which you wanted to count the number of STOCK# in the entries retrieved. The report might look like:

```
>REPORT
>>H1,"DIFFERENT STOCK# COUNT:",30
>>TF,STOCK-DETAIL.STOCK#,50,COUNT
>>END
```

This report would produce the following:

```
DIFFERENT STOCK# COUNT: 7
```

This count is incorrect if you wanted a count of the unique STOCK# entries. You need to use the NOREPEAT option on the TF statement. This tells QUERY to count only the first occurrence of each retrieved entry from STOCK-DETAIL. The report and its output are shown below:

```
>REPORT
>>H1,"DIFFERENT STOCK# COUNT:",30
>>TF,STOCK-DETAIL.STOCK#,50,COUNT,NOREPEAT
>>END

DIFFERENT STOCK# COUNT: 3
```

REPORT - REGISTER STATEMENTS

A register statement specifies an operation to be executed in one of 30 QUERY registers. The register statements are executed sequentially as they appear in the REPORT command, once for each data entry in the report (that is, once for each entry selected by the last retrieval command).

Syntax

$$\texttt{R}number,\ \left\{\begin{array}{l}\texttt{L[OAD]}\\ \texttt{A[DD]}\\ \texttt{S[UBTRACT]}\\ \texttt{M[ULTIPLY]}\\ \texttt{D[IVIDE]}\end{array}\right\},data\ element\ [\ \texttt{,NOREPEAT}\ ]$$

For example:

```
R3,ADD,PRICE
```

Where *number* = 3 and *data element* = PRICE

```
R0,M,"25"
```

Where *number* = 0 and *data element* = "25"

```
R5,DIV,R6
```

Where *number* = 5 and *data element* = R6

Parameters

number is an integer from 0 to 29 which identifies the register you want to use.

data element (1) is the name of a simple data item or a compound data item with an optional *subscript* parameter. Data items may be qualified with the data base and data set which they belong. The form is:

$$[\ data\ base\ name:\ ]\left[\begin{array}{l}data\ set\ name.\\ dummy\ data\ set\ name.\end{array}\right]\ data\ item\ name\ [\ (subscript)\ ]$$

dummy data set name is a temporary data set name used in multiple data set access. (Refer to the JOIN command.)

subscript is a number indicating which sub-item to access. *Subscript* is entered in parenthesis and must be an integer >= 1, and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

(2) is the contents of a register (R*n*).

(3) is a number enclosed in quotation marks (a numeric literal). You must specify the quotation marks, but QUERY will strip them and use the number in the operation.

`LOAD` replaces the current contents of the register with the *data element*.

`ADD` adds the *data element* to the contents of the register.

SUBTRACT	subtracts the *data element* from the contents of the register.
MULTIPLY	multiplies the contents of the register by the *data element*.
DIVIDE	divides the contents of the register by the *data element*.
NOREPEAT	can only be used if the REPORT command follows a MULTIFIND command. When NOREPEAT is used, the *data element* must be a data item name. The NOREPEAT option allows the total statement to be executed for a compound entry only if the simple entry has not been previously encountered in the statement. By default, the register statement is executed for all compound entries. The number of registers which use the NOREPEAT option, plus the number of sort levels for which the TOTAL statement uses NOREPEAT, must be less than or equal to 10. Refer to the JOIN command for information about compound data sets. For a more detailed explanation of this option refer to the section on REPORT Total Statements.

Discussion

After each operation is executed, the result is placed in the register specified at the beginning of the statement. For example, assume R2 contains 3 and R4 contains 2. After the statement below is executed:

```
R2,MULTIPLY,R4
```

R2 will contain 6.

Using QUERY Registers

The register statements in a REPORT command describe a fixed sequence of operations to be performed each time a new data entry is processed for the report. If you are familiar with programming techniques, you can consider the R*n* statements as a program "loop" which is executed once for each entry in the report.

R*n* statement execution affects only the 30 QUERY registers. No output results from the statements, although you can print the content of any register by using other REPORT command statements (with the exception of sort and header statements).

All register operations except LOAD are cumulative. When each R*n* statement is executed, the current contents of the register are operated on by the data element and the result is stored in the register again.

Initializing Registers

Each register is initialized to zero when the REPORT command begins execution. You can reset the register to zero in three ways:

1. Load a zero into the register. For example:

   ```
   R3,L,"0"
   ```

2. Use a total statement to reset the register to zero when a control break occurs. For example, the following statement sets Register 3 to zero when a control break occurs as a result of sort level 2.

   ```
   T2,R3
   ```

3. Use an arithmetic operation that results in zero. For example:

```
R4,SUBTRACT,QUANTITY (where QUANTITY is equal to the contents of R4)
```

If you divide the contents of a register by zero, the result is zero. Results of integer division are truncated.

Register and Data Types

Only numeric type data can be used in register operations. The following IMAGE data item types are allowed:

- whole numbers or integers (I1,I2,I4,J1,J2,J4,K1,K2,Z*n*, and P*n*)
- real numbers (R2)
- extended precision real numbers (R4).

The maximum length of P and Z data types that a register statement can handle is 20.

You can mix data types in a register operation. For example, you can add an integer to a real number. QUERY determines the data type of the register content by assigning an order of precedence to the data types, as follows:

```
HIGHEST    R4  (Extended precision type data)
           R2  (Real type data)

LOWEST     I1,I2,I4,J1,J2,J4,K1,K2,Zn,Pn
           (Integers and packed decimal numbers)
```

The new register content always has a data type which is the higher of the two operand types: the old register content or the *data element*. For example, if Register 2 is loaded with an integer and then multiplied by a real type data item, the content of R2 will be type real.

```
R2,L,"3"

R2,M,PERCENT    (where the value of PERCENT is 25.6)
```

All registers start with a default type of packed decimal. When you issue the REPORT command, QUERY cycles through all register statements (before executing the command) and assigns each register the highest type that the register will hold during the report. This ensures the correct typing of a register, in case its type changes during the report. For example:

```
R1,LOAD,DECITEM          Register 1 is type packed decimal      R2,LOAD,REALITEM
Register 2 is type real (R2)      R1,ADD,R2              Register 1 becomes type real (R2)
```

Because Register 1 is added with Register 2 later in the report, it is typed as real, instead of the initial default of packed decimal. When the first LOAD into Register 1 is performed, DECITEM will be converted into a real and loaded into Register 1. If Register 1 is printed with a detail statement, it will be displayed as a R2 data type.

You can convert a register to type R4 either by loading it (or by performing an arithmetic operation on it) with a data item of type R4. You can also convert a register to R4 by loading it with a real numeric literal that has more than 6 digits. Zeros preceding the decimal point or the most significant digit will be ignored, except in the case of all zeros. For example, all of the following convert Register 1 into an R4 data type:

```
R1,ADD,"000000.0"          (produces 0.0)
R1,LOAD,"0.000000"         (produces 0.0)
R1,MUL,"1.000000"          (produces 1.0)
R1,ADD,"500.0000"          (produces 500.0)
R1,LOAD,".0000100"         (produces .00001)
```

The following will not perform the conversion since leading zeros will not affect the output.

```
R1,LOAD,"000005.0"         (produces an R2 data type)
R1,LOAD,"000000.5"         (produces an R2 data type)
```

However, the following will perform the conversion.

```
R1,LOAD,"5.000000"         (produces an R4 data type)
R1,LOAD,".5000000"         (produces an R4 data type)
```

The largest integer (including all IMAGE integer data types) which a register can contain is 19 digits.

Real and extended precision numbers have the same limits in registers as IMAGE R2 and R4 data items. R2 can have 6 to 7 significant digits and R4 can have 16 to 17 significant digits. If a register calculation results in overflow, a message is printed on the $STDLIST device.

When mixing data types in arithmetic register computations, you should think about the order of precedence and its effect on the calculations. For example, if you operate on a real register number with an integer having 12 significant digits, the result will have 6 to 7 significant digits.

Numeric Literals

To use a constant number in a register operation, you enter the number surrounded by quotation marks (for example, "325", ".0013", "-3E-6"). This type of number is called a numeric literal.

Integer numeric literals can have at most 19 digits. The length of real numeric literals is limited only by the line length (or input record length). Limits for real numeric literal values and significant digits are the R2 limits. Numeric literals can contain the following characters within the quotation marks.

- the digits 0 through 9 (integer and real)
- the plus (+) and minus (-) signs (integer and real)
- the letter E, upper or lower case (real only)
- the decimal point (real only).

Embedded blanks in numeric literals are not accepted in register statements.

```
>REPORT
>>H1,"AS OF:",6
>>H1,DATE,15
>>H1,PAGENO,71
>>H1,"PAGE",69
>>H2,"BOBO'S MERCANTILE",45
>>H3,"ON HAND INVENTORY",45,SPACE A2
>>H7,"BIN#",4
>>H7,"SUPPLIER",14
>>H7,"STOCK",33
>>H7,"SHIP DATE",49
```

```
>>H7,"INVENTORY",68
>>H8,"AMOUNT",68
>>D1,STOCK#,36
>>D1,LASTSHIPDATE,48
>>D1,R12,68,E1
>>E2,"XX/XX/XX"
>>E1,"$$$,$$$,$$$,$$$.99"
>>S2,BINNUM
>>S1,SUPPLIER
>>S,LASTSHIPDATE
>>R12,LOAD,ONHANDQTY
>>R12,MULT,UNIT-COST
>>R8,ADD,R12
>>T2,R8,68,E1,SPACE B
>>G2,BINNUM,3,SPACE B
>>G1,SUPPLIER,20
>>T2," *",70
>>T2,R8
>>T2,"BIN TOTAL",14,SPACE B,SPACE A
>>TF,"TOTAL INVENTORY",20,SPACE B3
>>TF," **",71
>>TF,R9,68,E1,SKIP A
>>R9,ADD,R12
>>END
```

In the example above, a detail statement is added to print the content of Register 12 and edit with E1. Three register statements are added to load the value of ONHANDQTY into R12, multiply by UNIT-COST, and add the result to the contents of R8.

Three more total statements are added to print the content of Register 8 at each level 2 control break, to clear Register 8, and to print the content of Register 9 on the final total line. The last register statement adds R12 to R9 each time the data of another entry is printed in the report.

In this example, all register statements are executed every time a new entry is processed by the REPORT command.

```
AS OF: 01/07/86                                              PAGE 1
                          BOBO'S MERCANTILE
                          ON HAND INVENTORY


BIN#  SUPPLIER              STOCK       SHIP DATE          INVENTORY
                                                              AMOUNT
                     ONHANDQTY multiplied by UNIT-COST ------
                                                              |
  0   H & S SURPLUS                                           v
                          7391Z22F     8/13/85            $5,012.50
                          5405T14F     9/11/85           $12,129.60
                          6650D22S    12/17/85           $14,985.00

      BIN TOTAL                                           $32,127.10
                                                            ^
                                accumulated R12 values in R8    |
                                                R8 set to 0 ---
  1   ACME WIDGET
                          2457A11C    12/01/85      $553,477,666.95
      BAY PAPER CO.
                          7391Z22F    12/01/85            $4,704.00
      CARDINAL MILLS
                          5405T14F    11/28/85            $1,396.00
```

```
    JAKE'S JUNK
                          3739A14F    12/15/85          $1,189.32

    BIN TOTAL                                       $553,485,956.27  *


2   ACME WIDGET
                          4397D13P     3/02/85          $55,080.00
    CARDINAL MILLS
                          3586T14Y    11/20/85             $358.56

    BIN TOTAL                                            $55,438.56  *


3   ACME WIDGET
                          6650D22S    12/03/85          $75,716.62
    H & S SURPLUS
                          6650D22S    12/17/85             $187.85
                          6650D22S    12/15/85             $153.45

    BIN TOTAL                                            $76,057.92


    TOTAL INVENTORY       accumulated R12 values--->$553,248,579.85 **
```

REPORT - OUTPUT CONTROL STATEMENTS

Output control statements can be included in a REPORT command to alter the standard parameters for report output. There are five output control statements.

FORM	PURPOSE
LINES=*integer*	Specifies the number of lines per report page. *integer* can be between 10 and 32,767. If *integer* is 0, the page size is infinite.
NOPAGE	Suppresses page advancing at the beginning of each page (no margins are provided at the top and bottom of each page). All SKIP options are ignored, and all SPACE options are performed unconditionally.
[OUT=]LP	Sends the report output to the QSLIST device. Applies only to the current report. Refer to the OUTPUT= command for more information on QSLIST.
PAUSE	Causes the report output to pause after each page completes. Press (RETURN) to continue. PAUSE adds an extra line as (RETURN) is pressed. PAUSE is ignored in job mode or if output is sent to QSLIST.
UNIFYDETAIL	Causes page eject when a particular block of detail statements would split between two pages. A block is all the detail statements for each entry or compound entry.
NOPUNCH	causes negative values of type P, Z, I, J, and K to be printed without the overpunch characters. This statement only affects the output of header, detail, group, and total statements. This statement has no effect when used in the REPORT ALL command. Refer to the REPORT ALL command for a discussion of overpunch characters.

The standard parameters for report output are:

- 60 lines per page.
- Page advancing at the beginning of each report page. On the terminal, the page advancing appears as 6 line feeds.
- Output printed on the $STDLIST device (the terminal in session mode and line printer in job mode).
- No pauses while the report is being printed.

If you are using the REPORT ALL command, these statements must precede the keyword ALL.

Example

In the following example, entries are located with the FIND command. The REPORT ALL command prints entries without output control statements. The report is terminated with (CONTROL) Y. When the NOPAGE output control statement is used, QUERY does not skip lines for a top of page margin.

```
>F ALL LAST-NAME
USING SERIAL READ
13  ENTRIES QUALIFIED
>REPORT ALL


ACCOUNT          =54283540
LAST-NAME        =CORCORAN
FIRST-NAME       =CLIFFORD
INIT
(CONTROL) Y
< CONTROL Y >

>REPORT NOPAGE;ALL
ACCOUNT          =5428340
LAST-NAME        =CORCORAN
FIRST-NAME       =CLIFFORD
INITIAL          =C
    .                .
    .                .
    .                .
```

In the example below, QUERY pauses after 10 lines and (RETURN) must be pressed to continue the listing:

```
>REPORT
>>D1,LAST-NAME,20,SPACE A2
>>PAUSE
>>LINES=10
>>END
```

In the next example, the UNIFYDETAIL statement causes an entire block of detail lines to be printed together on one page.

```
>REPORT
>>UNIFYDETAIL
>>D1,ITEM1,5
>>D2,ITEM2,5
>>D3,ITEM3,15
>>S1,ITEM1
```

```
>>G1,"GROUP TITLE",11
>>END
```

REPORT ALL

Prints data item values of entries located by the last retrieval command without formatting.

Syntax

R[EPORT][*output control statement;*]ALL [,*character*]

For example:

```
>REPORT ALL,X
```

Where *character* = X

Parameters

output control statements	alter the standard output parameters. (Refer to the section on Output Control Statements under the REPORT command.)
character	is any printable ASCII character, except a minus sign (-). When *character* is used, the data item names are omitted from the report. In addition, negative values of type P or Z and positive values of type Z are printed with an overpunch character.
-	is a minus sign. When a minus sign is used, the data item names are printed with the data item values. In addition, QUERY edits negative values of type P or Z and positive values of type P to include the value sign. If the minus sign (-) not is used, negative values of type P or Z and positive values of type Z are printed with an overpunch character. Refer to the next page for more information on overpunch.

Discussion

The REPORT command prints a report of the data entries located by the last FIND, MULTIFIND, or SUBSET command.

REPORT output can be directed to any desired output device through the MPE :FILE command and the QUERY OUTPUT= command. Refer to the OUTPUT= command for more information.

If you are using REPORT ALL on an item type of U or X with a length greater than 50 characters, QUERY prints the data item value in multiples of 50 characters.

If you are using the REPORT ALL command in session mode and output is defined as LP, QUERY leaves 2 blank lines at the end of the page (page default is 60 lines). Extra lines can be eliminated by using the NOPAGE output control statement.

For multiple data set access (refer to the JOIN and MULTIFIND commands), REPORT ALL prints the data base name and data set name for each data set entry contained in a compound entry.

The following table summarizes the REPORT ALL options:

Table 3-9. REPORT ALL Options

COMMAND	OVERPUNCH USED	DATA ITEM NAMES PRINTED
REPORT ALL	Y	Y
REPORT ALL, X	Y	N
REPORT ALL, -	N	Y

Overpunch Characters

If the minus option is not used, QUERY prints negative values of type P or Z and positive values of type Z with a special character in the rightmost digit substituting for the minus or plus sign. This special character is called an overpunch character and varies according to the rightmost digit in the value. Table 3-10 shows the overpunch characters. For example, the number -104 is represented as 10M in QUERY.

Table 3-10. Overpunch Characters

RIGHTMOST DIGIT IN VALUE	POSITIVE REPRESENTATION	NEGATIVE REPRESENTATION
0	} (may vary with terminal)	} (may vary with terminal)
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

Table 3-11 shows the representation of P and Z type values which depends on how you entered the value and which REPORT ALL option, if any, you used. In this table, P4 and Z4 data types are shown. The output may vary for other lengths.

Table 3-11. Output of P and Z Type Values

COMMAND AND DATA TYPE	VALUE INPUT AS: +55	VALUE INPUT AS: 55	VALUE INPUT AS: -55
REPORT ALL			
P4 TYPE	ITEMNAME = 0055	ITEMNAME = 0055	ITEMNAME = 005N
Z4 TYPE	ITEMNAME = 005E	ITEMNAME = 0055	ITEMNAME = 005N
REPORT ALL, X			
P4 TYPE	0055	0055	005N
Z4 TYPE	005E	0055	005N
REPORT ALL, -			
P4 AND Z4 TYPES	ITEMNAME = 55	ITEMNAME = 55	ITEMNAME = -55

Examples

Example 1

```
>FIND ALL LAST-NAME
USING SERIAL READ
13  ENTRIES QUALIFIED
>REPORT ALL


ACCOUNT          =54283540
LAST-NAME        =CORCORAN
FIRST-NAME       =CLIFFORD
INITIAL          =C
STREET-ADDRESS   =6105 VALLEY GREEN DR.
CITY             =CARMEL
STATE            =CA
ZIP              =93921
 .                  .
 .                  .
 .                  .
```

As shown in the example above, once entries have been found, REPORT ALL prints the value for each item to which you have access.

Example 2

```
>REPORT ALL,X


54283540
CORCORAN
CLIFFORD
C
6105 VALLEY GREEN DR.
CARMEL
CA
93921
7.10000
```

```
54283545
MAYFIELD
WILLIAM
(CONTROL) Y
< CONTROL Y >
```

When REPORT ALL is used with a character, the data values are printed without the data item names and data set names. You can terminate the report at any time by entering (CONTROL) Y.

REPORT procedure

Executes a REPORT command stored as a procedure in the current Proc-file.

Syntax

R[EPORT][*output control statements*;]*procedure name* [,*character*]

For example:

```
>REPORT REP4,J
```

Where *procedure name* = REP4, *character* = J

```
>R FINDTAX
```

Where *procedure name* = FINDTAX

Parameters

output control statements	Refer to the section on Output Control Statements in the REPORT command.
procedure name	is the name of a REPORT command stored as a procedure in the current Proc-file.
character	is any printable ASCII character. If *character* is included in this command, the REPORT procedure is listed before it is executed.

Discussion

QUERY searches the current Proc-file and executes the REPORT command stored under the *procedure name*. The data entries used are those located by the previous retrieval command. If the procedure does not exist in the current Proc-file, or if the Proc-file has not been declared, QUERY prints an error message.

Before the procedure is executed, QUERY checks it for proper syntax. If any statement (except the first one) is incorrect, QUERY issues an error message and prompts you with >>. At this point, you can enter one or more statements to replace the statement causing the error. The statements must be on one line and separated by semicolons. If any of these substitute statements is in error, QUERY issues another error message and prompts you again. Once satisfactory statements have been entered, QUERY executes the procedure using the newly entered statements in place of the statements in error. If the first report statement is in error, the REPORT command terminates.

The corrected statements are not permanent. Once the procedure has executed, the entered statements are lost. The procedure remains unchanged in the Proc-file. To permanently change the procedure, you must use the ALTER command.

Figure 3-10, for example, shows a REPORT procedure named SHIPMNTS. The third line contains a sort statement with a comma missing.

```
PROCEDURE: REP4

001  REPORT
002  H1,"MONTHLY SHIPMENTS",25,SPACE A2
003  S1 STOCK,STOCK#
004  G1,STOCK#,15
005  D1,LASTSHIPDATE,25
006  END
```

Figure 3-10. REPORT Procedure Named SHIPMNTS

When the procedure is executed. QUERY prints the offending sort statement and prompts for a correction. Once you enter a correct statement or series of statements, execution of the procedure continues. For example you might respond with:

```
>REPORT REP4
003  S1 STOCK#
EXPECTED A ","
>>S1,STOCK#;S,LASTSHIPDATE
```

Not only has the sort statement been temporarily fixed, but another sort statement has been added as well. Once the procedure has executed, the corrections and additions are lost.

When using REPORT procedures, you can leave out END. When the procedure is executed, QUERY will prompt you for the missing END. You can then enter special sort or output control statements to vary the report and its output parameters. The added statements are lost once the procedure is executed.

SAVE

Creates an MPE file containing data from the the IMAGE data entries referenced in the current select file.

Syntax

SA[VE]*filename* [*/lockword*]

Parameters

filename	is the MPE filename by which the saved select file is known.
lockword	is the lockword that will be associated with the saved select file.

Discussion

The SAVE command creates a new MPE file containing the IMAGE data entries selected by the most recent retrieval command. This select file is saved as a self-describing (SD) file which means that its user label contains information about its contents in a standard format. Refer to Appendix E for further discussion on self-describing files.

If a file with the specified name already exists, the select file is not saved and the command terminates. If you terminate the command with a CONTROL Y, the existing file is not overwritten and the select file is not saved.

QUERY's internal select file consists only of record numbers. However, when you use the SAVE command, the data, from the entries that qualified when the retrieval command was executed, is copied into the file. This file can then be used by other subsystems such as DSG/3000. The select file and the data in the database remain unchanged by the SAVE command.

Saving Multiple Data Set Select Files

The combined length of the non-compound data entries forming a compound data entry cannot exceed 2047 words. The maximum number of data items that can be contained in a compound data entry is 510. To save a multiple data set select file, the previous JOIN command cannot contain any @ signs.

A data item name can belong to only one data set named in the JOIN command unless both of the following are true:

- The data set and data item are both named in a data item equivalence for all the data sets named in the JOIN command that contain that data item.
- The data item is not a compound item.

SETLOCKS

Prevents automatic unlocking of a data set.

Syntax

```
SET[LOCKS]
```

Discussion

The SETLOCKS command prevents the automatic unlocking of a data set after a retrieval, update, or REPORT command has been executed. Normally, QUERY locks a data set while these commands are executing, but not in the interim between commands. By issuing the SETLOCKS command, you can lock the data set during the time between commands. This ensures that retrieval information does not change before it is reported. Locking remains in effect until a RELEASE command is issued.

QUERY decides what data sets to lock based on the retrieval or updating command you use.

```
>SETLOCKS
>PROC=PROCA
>FIND SALES.STOCK#>20
USING SERIAL READ
6  ENTRIES QUALIFIED
WARNING:LOCKS ARE BEING HELD
>REPORT ALL
```

In the example above, locking takes place when the FIND command is entered and will remain in effect until a RELEASE command is entered. After a SETLOCKS command has been issued, QUERY provides a warning prompt following every command. If you enter a SETLOCKS command a second time you will receive the following message:

```
SETLOCKS COMMAND ALREADY IN EFFECT
```

Locks incurred by any QUERY command following a SETLOCKS command will remain in effect until a RELEASE or EXIT command is entered, or until QUERY is aborted in either job or session mode. The SETLOCKS command overrides LOCKOPTION=OFF until the RELEASE command is issued, at which point LOCKOPTION will take effect again with the same value it had prior to the SETLOCKS command.

Note The SETLOCKS command may impact other users of the data base(s). Locks should be released as soon as possible. Particular care should be taken when retrieving from multiple data sets or multiple data bases. Locking only takes place in modes 1, 2, and 5.

SHOW

Displays the current state of an aspect of the QUERY environment.

Syntax

```
SH[OW] { DBLIST [data base name]
       { JOIN [data base name]
       { LOCKOPTION [data base name]
       { LANG[UAGE]
```

Parameters

DBLIST	shows the contents of the data base list. (Refer to the DBLIST= command).
JOIN	shows the most recent valid JOIN command, if any.
LOCKOPTION	shows the current state of LOCKOPTION which can be reset with the ASSIGN command.
LANGUAGE	is the user language. (Refer to the LANGUAGE= command and Appendix D for more information.)
data base name	is the name of a data base opened with a MULTIDB command. If a name is not specified, QUERY will use the primary data base defined by the DEFINE or DATA-BASE= command.

```
>SHOW LOCKOPTION
LOCKOPTION=ON
>ASSIGN LOCKOPTION=OFF
>SHOW LOCKOPTION
LOCKOPTION=OFF
```

SUBSET

Retrieves entries from a select file.

Syntax

SUB[SET] [#LIMIT=*i*] { *relation* / *item identifier* M[ATCHING] "*pattern*" }
[{ AND / OR } { *relation* / *item identifer* M[ATCHING] "*pattern*" }] ... [END]

Parameters

i is an integer specifying the maximum number of qualifying entries you want to retrieve. When the #LIMIT parameter is specified, only the first *i* qualifying entries are placed in the select file. If fewer than *i* entries exist, then all entries that qualify are put in the select file.

item identifier takes the form:

[*data base name:*] [*data set name.* / *dummy data set name.*] *data item name*
[(*subscript*)]

data base name is the name of a data base specified in either the DEFINE, DATA-BASE=, or MULTIDB command.

data set name is the name of a data set in the current data base.

dummy data set name is a temporary data set name used in multiple data set access (refer to the JOIN command).

data item name is the name of a data item contained in the data base. If a *data set name* is used, the data item must belong to the specified data set. To use the MATCHING option, the data item must be type X or U.

subscript is a number indicating which sub-item to access. *Subscript* must be an integer >= 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

MATCHING allows you to retrieve data based on the comparison of data items with a specified "pattern". Refer to the FIND command for a detailed description of MATCHING.

"pattern" pattern must be enclosed in quotation marks. Refer to the FIND command for further specifications.

relation takes the form:

[*data base name:*] [*data set name.* / *dummy data set name.*] *data item name*
[(*subscript*)] *relop* { *"value"* / $MISSING } [, { "*value*" / $MISSING }] ...

data item name can be of any data type.

relop is a relational operator as shown in Table 3-10.

value is the data item value. It must be the same type and within the same value range as the data item named in the *relation*. *Value* need not be enclosed in quotation marks unless the value contains special characters. A *value* which is not contained in quotation marks is upshifted. For example, California is converted to CALIFORNIA before it is compared to data item values in the data base. *Value* must be an exact match for character type data items (type U and X). You can use null values. Refer to "Using Null Values" under the FIND command.

`$MISSING` is used to retrieve a missing (non-compound) entry from a compound data set. Refer to the MULTIFIND command for more information.

`END` must be included in a procedure.

Discussion

This command provides a specialized kind of retrieval by selecting subsets of previously retrieved entries. It operates in a similar manner to the FIND command except that it searches only entries already in the select file established by the most recent retrieval command.

All data items named in the SUBSET command must belong to the data set(s) accessed in the most recent FIND or MULTIFIND command.

Up to 50 logical connectors (AND,OR) can be used in a SUBSET command. Refer to "Logical Connectors" under the FIND command.

When the SUBSET command is entered, the current select file is saved as a temporary file which can be restored and made accessible through the use of the UNDO command. (Note: the SAVE command saves the current select file as a permanent file.) The temporary select file is then examined for the subset of entries you request. Those entries that qualify are placed in the new current select file. If you use the UNDO command, you will no longer have access to those entries retrieved by the most recent SUBSET command.

Table 3-12. Retrieval Command Relational Operators

OPERATOR	MEANING
= IS IE EQ	is equal to (Multiple *values* may be used with these operators.)
# <> ISNOT INE NE	is not equal to (Multiple *values* may be used with these operators.)
< ILT LT	is less than
>= INLT GE	is not less than (is greater than or equal to)
> IGT GT	is greater than
<= INGT LE	is not greater than (is less than or equal to)
IB $value_1$, $value_2$	is between (and including) $value_1$ and $value_2$
	Note: The operators <>, <=, and >= cannot have any intervening spaces (embedded blanks).

Example

```
>FIND PERIODICAL.JOURNAL="LIFE SCIENCES"
>SUBSET PERIODICAL.YEAR=85
>SAVE MFILE
```

In the example above, the SUBSET command only searches those entries that qualified from the FIND command. The SAVE command saves entries from the current select file as a permanent file named MFILE.

SUBSET procedure

Executes a SUBSET procedure stored in the current Proc-file.

Syntax

SUB[SET] *procedure name* [,*character*]

For example:

```
>SUBSET ACCT
```

Where *procedure name* = ACCT

```
>SUB USERS,X
```

Where *procedure name* = USERS, *character* = X

Parameters

procedure name	is the name of a SUBSET command previously stored as a procedure using the CREATE command. The procedure must exist in the current Proc-file, specified with the PROC-FILE = command or in response to the PROC-FILE prompt in the DEFINE command.
character	is any printable ASCII character. If *character* is included in the command, the SUBSET procedure is printed on the standard list device before executing.

Discussion

QUERY searches the current Proc-file and executes the procedure named in the command. If the Proc-file has not been declared, or the procedure does not exist in the Proc-file, or the procedure is incorrect in some way, you are informed by an error message.

If null data values appear in the procedure, QUERY prompts you for the necessary values. Refer to the FIND command for further discussion of null data values. For more information about storing and using SUBSET procedures, refer to the CREATE command.

TRANSBEGIN

Marks the beginning of a logical logging transaction.

Syntax

T[RANSBEGIN][*data base name:*][*"text"*]

Parameters

data base name	is the name of a currently open data base. If no data base name is specified, the transactions are logged against the data base opened with the DEFINE or DATA-BASE= command.
"text"	is an optional character string to be written to the log file. It can contain any information that you might want to include about the transaction.

Discussion

This command allows you to designate the beginning of a logical logging transaction from the QUERY subsystem in a similar manner to the IMAGE DBBEGIN intrinsic.

A transaction is defined as a sequence of one or more modifications which changes a data base from one consistent state to another. Logging allows you to keep a record of transactions in the event of a system failure. Marking the transaction with a TRANSBEGIN and TRANSEND, allows you to restore your data base to a consistent state by suppressing those transactions that failed to complete. For further information on transaction logging, refer to the *IMAGE Reference Manual*.

Logging must have already been enabled from the IMAGE subsystem for the TRANSBEGIN command to work. Consult your data base administrator about enabling the logging facility. If logging has not been enabled, you will receive the following message:

```
LOGGING NOT ENABLED FOR THIS USER
```

If you enter the TRANSBEGIN command twice without entering an intervening TRANSEND command, the following message is printed:

```
A TRANSACTION IS IN PROGRESS
TRANSEND MUST BE CALLED BEFORE ANOTHER TRANSBEGIN
```

Entering EXIT while transaction logging is in effect will not abort the logging process. Note, however, that when logging completes, QUERY performs an implicit TRANSEND, your data base is closed, and the subsystem is exited.

TRANSEND

Marks the end of a logical logging transaction.

Syntax

TRANSE[ND][*data base name:*][*"text"*]

Parameters

data base name	is the name of a currently open data base. If no data base name is specified, the transactions are logged against the data base opened with the DEFINE or DATA-BASE= command.
"text"	is an optional character string to be written to the log file. It can contain any information that you might want to include about the transaction.

Discussion

This command allows you to designate the end of a logging transaction from the QUERY subsystem in a similar manner to the IMAGE DBEND intrinsic.

A transaction is defined as a sequence of one or more modifications which changes a data base from one consistent state to another. Logging allows you to keep a record of transactions in the event of a system failure. Marking the transactions with a TRANSBEGIN and TRANSEND, enables you to restore your data base to a consistent state by suppressing those transactions that failed to complete. For further information on transaction logging, refer to the *IMAGE Reference Manual*.

Logging must be enabled, and a TRANSBEGIN entered before a TRANSEND command is accepted. If you enter a TRANSEND without having entered a prior TRANSBEGIN command, the following message is returned:

```
NO TRANSACTIONS IN PROGRESS
```

TRANSMEMO

Allows a message to be written to the log file.

Syntax

TRANSM[EMO] [*data base name:*] [*"text"*]

Parameters

data base name	is the name of a currently open data base. If no data base name is specified, the message is logged against the data base opened with the DEFINE or DATA-BASE= command.
"text"	is a character string to be written to the log file.

Discussion

This command allows you to write a message to the log file that adds additional information, or that identifies transactions in the event of a system failure and subsequent recovery. A message can be added at any time, but can be no more than 512 characters.

UNDO

Restores the previous select file.

Syntax

```
UN[DO]
```

Discussion

The UNDO command clears the last SUBSET retrieval and returns you to the previous level of data retrieval. Each group of retrieved entries is stored in a temporary internal file known as a select file. Entering the UNDO command removes the last group of retrieved entries (select file) and restores the previous group of retrieved entries as the current select file. The restored select file can be the result of a previous SUBSET, FIND, or MULTIFIND command, but UNDO must follow a SUBSET command. After each SUBSET retrieval, QUERY prints the number of qualified entries retrieved.

Examples

Example 1

```
--> >FIND ALL PERIODICAL.JOURNAL
|   USING SERIAL READ
|   60 ENTRIES QUALIFIED
|   >SUBSET PERIODICAL.JOURNAL = CALIFORNIA
|   30 ENTRIES QUALIFIED
--  >UNDO
```

In this example, the previous 60 entries are in the current select file again.

Example 2

```
     >FIND ALL PERIODICAL.JOURNAL
     USING SERIAL READ
     60 ENTRIES QUALIFIED
--> >SUBSET PERIODICAL.JOURNAL = CALIFORNIA
|    30 ENTRIES QUALIFIED
|    >SUBSET PERIODICAL.YEAR = 85
|    10 ENTRIES QUALIFIED
---  >UNDO
```

In example 2, the previous 30 entries are in the current select file again.

UPDATE ADD

Adds data entries to a manual master or detail data set.

Syntax

$$\left\{\begin{array}{l}\texttt{U[PDATE]ADD,}\\ \texttt{AD[D]}\end{array}\right\}\texttt{[}\textit{data base name:}\texttt{]}\textit{data set name}$$

For example:

```
>UPDATE ADD,LABOR
```

Where *data set name* = LABOR

```
>AD PAYROLL
```

Where *data set name* = PAYROLL

Parameters

data base name	is the name of a data base specified in either the DEFINE, DATA BASE=, or MULTIDB command.
data set name	is the name of a data set in the data base. The data set must be either a manual master or detail. You must have write access as determined by the password you enter in response to the PASSWORD= prompt or through the PASSWORD= command.

Discussion

When you use the UPDATE ADD command, QUERY prompts you for data item values by printing the names of the data items. You type the desired value following the data item name. QUERY will continue to prompt you for data item values until you terminate the command.

In a procedure, you must use the UPDATE ADD form of the command. UPDATE can be abbreviated. ADD cannot be abbreviated when it is used with the UPDATE keyword.

You must have specified mode 1, 3, or 4 to use this command.

Example

```
>AD PRODUCT
STOCK#          =>>12345678
DESCRIPTION     =>>"ANTIMACASSAR"

STOCK#          =>>(RETURN)
STOCK#          =>>54231234
DESCRIPTION     =>><(RETURN)

STOCK#          =>>//
>
```

In the example above, the STOCK# is a search item. A value must be provided. Since DESCRIPTION is not a search item, it can be left blank by pressing the (RETURN) key.

However, if you try to use RETURN for a search item, QUERY will prompt you again for the value. The command is terminated with two slashes.

If you are in session mode and enter a value that is too large or that contains invalid characters, QUERY issues an error message and reprompts with the same data item name.

If illegal data is encountered while QUERY is running in a job or from an XEQ file, QUERY will issue an error message and stop adding data. In job mode, QUERY will also terminate.

The UPDATE ADD command will prompt you for each sub-item (each occurrence of a compound data item). If you want to terminate the prompting for sub-items, enter two asterisks (**). QUERY will then prompt you for the next item.

```
>UPDATE ADD,SALES
SALESPERSON      =>>JONES
MONTHLY-SALES(1)=>>450
MONTHLY-SALES(2)=>>700
MONTHLY-SALES(3)=>>**
REGION           =>>MIDWEST

SALESPERSON      =>>//
```

Using Quotation Marks

Values can be entered with, or without quotation marks. When you enter U or X type values without quotation marks, leading blanks are ignored and the value is left-justified in the data item field. Blanks or any other special characters are entered in the data field only if the value is entered with quotation marks. For example, if F-NAME is a U type data item, then:

```
F-NAME       >>  SALLY
```

is entered in an 8-character field as:

```
SALLY
```

```
F-NAME       >>"  SALLY"
```

is entered an 8-character field with 2 leading blanks.

A U type data item is automatically upshifted if it is entered in lowercase. You can override the upshifting by enclosing the entry in quotation marks. In the next two examples, F-NAME is a U type data value.

```
F-NAME       >>sally
```

is entered as:

```
SALLY
```

```
F-NAME       >>"sally"
```

is entered as:

```
sally
```

If quotation marks are to be included as part of the value to be entered, they must be entered twice to avoid being confused with delimiters. For example, the value:

```
ROBERT "BOB" BRUN
```

must be entered as:

```
"ROBERT ""BOB"" BRUN"
```

Numeric type values (all types except U and X) are always right-justified whether entered with or without quotation marks. Numbers entered in X type data items are always left-justified.

Null Values

Only search and sort items must have values supplied. If you do not want to enter a value for other items, you can enter a null value. In session mode press (RETURN) without any preceding characters. In job mode, supply a blank line to indicate a null value. Numeric data items which do not contain a value are set to zero, character items are set to blanks.

Real Number Values

Real number values can be entered as either fixed or floating point numbers. An example of a fixed point number is:

```
23.45
```

An example of a floating point number is:

```
2.345E+01
```

The signed integer following the E stands for a power of 10 to be multiplied by the number to the left of the E. Both examples above stand for the same number.

Detail Data Sets

If a detail data set is linked to one or more *manual* master data sets, you must know the content of the master set before you can add entries to the detail data set. Each detail entry is associated with a master entry through the same search item. The detail search item and the master search item must have the same value, and the value must exist in a master entry before it can be added to a detail data entry. If you enter a search item value that does not exist in the master data set linked to the detail set through that search item, QUERY prints the following error message:

```
MISSING SEARCH KEY VALUE FOR data item name IN MASTER data set name

==========|ENTRY CANNOT BE ADDED|==========
```

Terminating UPDATE ADD

Once you have supplied values (null or otherwise) for all the data items in the entry, QUERY begins prompting you for another entry. To continue adding entries, merely enter an appropriate value at the prompt. To terminate UPDATE ADD command, either enter two slashes (//) followed by (RETURN) or enter (CONTROL) Y.

The command can be terminated at any time. However, if you want to save the values you have entered for the current entry, do not use the (CONTROL) Y or two slashes before you have been prompted for the last value of the current entry. Normally you will terminate the command in response to the first data item prompt in an entry.

Example

In the example below, the user tries to add an account to the SALES detail data set, but the account is not already in the CUSTOMER master data set. QUERY prints an error message and reprompts for data item values. The command is terminated with two slashes. Next, the account is added to the CUSTOMER master data set. The original entry can then be added to the SALES detail data set. Note that dates are automatically added to the DATE-MASTER automatic master data set.

```
>ADD SALES
ACCOUNT          =>>86110047
STOCK#           =>>14519990
QUANTITY         =>>6
PRICE            =>>2000
TAX              =>>RETURN
TOTAL            =>>RETURN
PURCH-DATE       =>>121585
DELIV-DATE       =>>121685
MISSING SEARCH KEY VALUE FOR ACCOUNT IN MASTER CUSTOMER

==========|ENTRY CANNOT BE ADDED|===========

ACCOUNT          =>>//
>ADD CUSTOMER
ACCOUNT          =>>86110047
LAST-NAME        =>>CELERY
FIRST-NAME       =>>ALLISON
INITIAL          =>>B.
STREET-ADDRESS   =>>18 ASCOTT AVE.
CITY             =>>CARMEL
STATE            =>>CA
ZIP              =>>93921
CREDIT-RATING    =>>1.2


ACCOUNT          =>>//
>ADD SALES
ACCOUNT#         =>>86110047
STOCK#           =>>141990
QUANTITY         =>>6
PRICE            =>>2000
TAX              =>>RETURN
TOTAL            =>>RETURN
PURCH-DATE       =>>121585
DELIV-DATE       =>>121685


ACCOUNT          =>>//
```

UPDATE DELETE

Deletes data entries from a data set.

Syntax

```
{U[PDATE]DELETE }
{DEL[ETE]       }
```

Discussion

This command is an extension of a retrieval command in that it deletes those entries selected by the previous retrieval command. All of the entries must reside in the same data set. UPDATE DELETE may only be used following the FIND or FIND ALL command or SUBSET command following a FIND or FIND ALL command. Entering FIND CHAIN or MULTIFIND (selecting entries from more than one data set) and then UPDATE DELETE is not allowed.

To delete entries from a data set, you must have write access to the data set, as determined by the password you enter in response to the PASSWORD= prompt or through the PASSWORD= command.

When you enter an UPDATE DELETE command, QUERY prints the message:

```
DELETE ALL RETRIEVED ENTRIES (YES OR NO)?
```

This message is reminder that all the entries selected by the last retrieval command will be deleted by the command. If you respond YES, QUERY deletes the entries. If you respond NO, the command is ignored and you are prompted for another command. In job mode, the entries are deleted without confirmation.

In a procedure, you must use the UPDATE DELETE form of the command. UPDATE may be abbreviated. DELETE may not be abbreviated when using it with the UPDATE keyword.

You must have specified a mode of 1, 3, or 4 in order to use this command.

Master Data Entries

QUERY disallows any attempt to delete the master entry if its search item value still exists in the search items of the appropriately linked detail data sets. However, QUERY will continue to delete as many entries as it can. When all possible entries have been deleted, QUERY will issue a message indicating how many master entries with associated details were encountered, and how many entries were able to be successfully deleted.

```
>F CUSTOMER.ACCOUNT IS 07954001
1  ENTRIES QUALIFIED
>U DELETE
DELETE ALL RETRIEVED ENTRIES (YES OR NO)?
>>YES
1 ENTRIES NOT DELETED BECAUSE CORRESPONDING DETAIL DATA ENTRIES EXIST
0 OF 1 QUALIFYING ENTRIES DELETED

>FIND SALES.ACCOUNT IS 07954001
1 ENTRIES QUALIFIED
>DEL
DELETE ALL RETRIEVED ENTRIES (YES OR NO) ?
>>YES
```

```
>
```

In this example, there is a detail data entry in SALES with an ACCOUNT value of 07954001, so QUERY will not delete the entry from the CUSTOMER master data set.

Next, an entry with an ACCOUNT equal to 07954001 is deleted from the SALES detail data set. Now the ACCOUNT equal to 07954001 in the CUSTOMER master data set can be deleted.

UPDATE REPLACE

Modifies the value of data items.

Syntax

```
{ U[PDATE]REPLACE, }
{ REPL[ACE],       } data item name [(subscript)]= "value";
[data item name [(subscript)]= "value"; ] ... END
```

For example:

```
>UPDATE REPLACE,DATE="741010";END
```

Where *data item name* = DATE and *value* = "741010"''

Parameters

data item name	is the name of a simple data item, or the name of a compound data item with an optional *subscript* parameter, contained in the entries selected by the last FIND command.
"value"	is the new value for the data item surrounded by quotation marks. It must be the proper size and type for the named item. Quotation marks are required. QUERY prints an error message if values of improper size or type are entered.
subscript	is the number indicating which sub-item you want to replace. *Subscript* is entered with parenthesis and must be an integer > 1 and <= the number of sub-items defined for the compound item. QUERY will default to the first sub-item if no *subscript* is specified.

Discussion

UPDATE REPLACE is an extension of the retrieval commands in that it operates on data entries selected by the previous retrieval command. All of the data entries selected must be from the same set. UPDATE REPLACE can only be used following a FIND or FIND ALL command or a SUBSET command following a FIND or FIND ALL command. Entering a FIND CHAIN or MULTIFIND command (which selects entries from a detail set and one or more master data sets) followed by an UPDATE REPLACE command is not allowed.

You must have specified a mode of 1, 2, 3, or 4 to use this command.

QUERY does not allow you to modify items in the data base defined as search or sort items.

The UPDATE REPLACE command consists of a series of data item name/data item value pairs separated by semicolons. When the command is entered, QUERY replaces the value of each data item named in the command with the new value enclosed in quotation marks. The names of data items appearing in the command must belong to the data entries selected by the last retrieval command.

In a procedure, you must use the UPDATE REPLACE form of the command. UPDATE can be abbreviated. REPLACE cannot be abbreviated when used with the UPDATE keyword.

If an IMAGE error occurs during the replacement of data entries, QUERY issues the following message to indicate how many entries were successfully updated before the error occurred:

```
X OF Y ENTRIES REPLACED
```

X is the number of entries that were successfully updated, and Y is number of retrieved entries.

Session Mode

When you enter an UPDATE REPLACE command from a terminal, QUERY checks each line of the command as it is entered. If an error occurs, the line is ignored, an error message is sent, and you are prompted for another line. QUERY continues to prompt for lines until you enter END. END is required.

The replacement statements (data item name/data item value pairs) can be entered on one line:

```
>UPDATE REPLACE,DATE="851010";HOURS="4";END
```

or on several lines:

```
>UPDATE REPLACE,
>>DATE="851010";
>>HOURS="4";
>>END
>
```

Job Mode

If you are entering the command in job mode, the replacement statements can appear in one record or several consecutive records. If an error occurs, either illegal data or an IMAGE error, QUERY will terminate.

Null Values

You can store an UPDATE REPLACE command as a procedure with null values for the data items. You do this by entering a pair of quotation marks with no intervening characters *data item name*= (for example, DATE="").

When such a procedure is executed using the UPDATE *procedure name* command, QUERY prompts you for a value by printing the message:

```
WHAT IS THE VALUE OF data item name
>>
```

Where *data item name* is the name of the data item associated with the null value in the UPDATE REPLACE command. The desired value must be entered without quotation marks. All characters entered (including blanks, quotation marks, and other characters) are significant. The maximum number of characters which can be entered as a value is 72. Lowercase characters are upshifted unless the data item is type X.

Examples

Example 1

```
>F CUSTOMER.FIRST-NAME=GENEVA
USING SERIAL READ
1  ENTRIES QUALIFIED
>REPL,STREET-ADDRESS="868 DOYLE ROAD";
>>END
>LIST CUSTOMER FOR FIRST NAME=GENEVA

   ACCOUNT  LAST-NAME         FIRST-NAME  IN   STREET-ADDRESS

  10034764  SLATER            GENEVA           868 DOYLE ROAD
```

In example 1, the CUSTOMER entry with FIRST-NAME of GENEVA is located. The street address is changed. The LIST command is used to check the results.

Example 2

```
>AD SALES
ACCOUNT          =>>67795400
STOCK#           =>>1169331
QUANTITY         =>>6
PRICE            =>>1800
TAX              =>>RETURN
TOTAL            =>>RETURN
PURCH-DATE       =>>121585
DELIV-DATE       =>>121685

ACCOUNT          =>>//
>F SALES.ACCOUNT IS 67795400
1  ENTRIES QUALIFIED
>UPDATE REPLACE,
>>TAX="25";
>>TOTAL="1825";
>>END
```

The UPDATE REPLACE command can also be used to fill in blank values. In the second example, no values are entered for TAX and TOTAL when the entry is added. The UPDATE REPLACE command is then used to add these values, after the entry has been located with the FIND command.

Example 3

```
>FIND PRODUCT.STOCK# IGT 70000000
USING SERIAL READ
3  ENTRIES QUALIFIED
>UPDATE REPLACE,
>>DESCRIPTION="OBSOLETE";
>>END

>REPORT ALL

STOCK#           =7931Z22F
DESCRIPTION      =OBSOLETE

STOCK#           =78787878
DESCRIPTION      =OBSOLETE

STOCK#           =98989898
DESCRIPTION      =OBSOLETE
```

In example 3, REPORT ALL is used to verify the replacements. The entries are still available from the last FIND. Each PRODUCT data set entry with a STOCK# greater than 70000000 is changed so that the description is OBSOLETE.

UPDATE procedure

Executes an UPDATE command stored as a procedure in the current Proc-file.

Syntax

U[PDATE] *procedure name* [,*character*]

For example:

```
>UPDATE UPDNAMES
```

Where *procedure name* = UPDNAMES

```
>U CHANGE,%
```

Where *procedure name* = CHANGE and *character* = %

Parameters

procedure name	is the name of an UPDATE ADD, UPDATE REPLACE, or UPDATE DELETE command procedure in the current Proc-file.
character	is any printable ASCII character.

Discussion

You can use the CREATE command to store any of the three forms of the UPDATE command. Once the procedure is stored in the Proc-file, you use this form of the UPDATE command to execute it. Refer to the CREATE command for information on creating a procedure.

When you execute the UPDATE procedure, QUERY first checks each line of the procedure. If an error occurs, a message is sent and the incorrect statement in the line is ignored. All other correct statements in the procedure are executed. If the *character* is included in the command, the procedure is listed before the execution begins.

Example

```
>>F LAST-NAME IS MAYFIELD
USING SERIAL READ
1 ENTRIES QUALIFIED
>PROC-FILE  =PROCX
>UPDATE CHANGE
WHAT IS THE VALUE OF THE CITY
>>PETALUMA
WHAT IS THE VALUE OF THE STREET-ADDRESS
>>37 41ST AVE.
WHAT IS THE VALUE OF THE CREDIT RATING
>>8.5
```

In this example, the FIND command is used to locate the entry to be updated. The procedure named CHANGE (located in Proc-file PROCX) contains the following:

```
U REPLACE,
CITY="";
STREET-ADDRESS="";
CREDIT-RATING="";
```

```
END
```

When using null values, QUERY prompts for values when the procedure is executed.

VERSION

Displays the current version of QUERY and IMAGE procedures and utilities.

Syntax

V[ERSION]

Discussion

This command will print the current version, update, and fix level information for QUERY and all IMAGE procedures and program files on the standard list device. If OUTPUT=LP has been specified previously, the information is also printed on the QSLIST device. (Refer to the OUTPUT= command for a description of QSLIST.)

```
>VERSION

     QUERY       C.00.08

IMAGE PROCEDURES:
     DBOPEN      C.0A.16
     DBINFO      C.00.17
     DBCLOSE     C.00.12
     DBFIND      C.00.15
     DBGET       C.00.15
     DBUPDATE    C.00.15
     DBPUT       C.00.15
     DBDELETE    C.00.15
     DBLOCK      C.00.16
     DBUNLOCK    C.00.16
     BIMAGE      C.00.16
IMAGE PROGRAM FILES
     DBSCHEMA.PUB.SYS  C.00.10
     DBSTORE.PUB.SYS   C.00.00
     DBRESTORE.PUB.SYS C.00.00
     DBUNLOAD.PUB.SYS  C.00.00
     DBLOAD.PUB.SYS    C.00.00
     DBUTIL.PUB.SYS    C.00.17
```

XEQ

Executes QUERY commands from a file instead of the standard input device.

Syntax

X[EQ]*filename* [,NODATA]

For example:

```
XEQ CFILE,NODATA
```

Where *filename* = FILE and *option* = ,NODATA

Parameters

filename	is the name of an ASCII file containing commands and parameters.
NODATA	if specified, QUERY prompts for null data values and any command parameters when the XEQ file is executed.

Discussion

When the XEQ command is entered, QUERY reads the specified file and executes the commands until an end-of-file or another XEQ command is encountered. Any command input is also read from *filename* unless NODATA is specified. When an end-of-file is reached, control returns to the original command input device (in session mode, the terminal, or in job mode, the job input device).

Only the first 72 characters of each record are read. You can continue a command on the next record by entering an ampersand (&) as the last non-blank character in the current record.

If an error occurs while opening the data base in session mode, QUERY will close the XEQ file and prompt for another command. In job mode, QUERY closes the XEQ file and the job is terminated.

An XEQ command within another XEQ file will close the first file and open the new one. The initial file is never reopened. QUERY performs an end-of-file on the first file and control transfers to the new file.

The NODATA option specifies that the XEQ file only contains commands and does not contain any data needed by the command. QUERY will prompt you for the data.

Control Y

When you enter (CONTROL) Y during the execution of an XEQ file, QUERY will terminate the command currently executing, close the XEQ file and prompt for another command.

If a FIND, MULTIFIND, or SUBSET command is executing when (CONTROL) Y is entered, QUERY will print the following message.

```
x ENTRIES HAVE QUALIFIED, DO YOU WANT TO CONTINUE SEARCHING?
```

If you respond YES, QUERY will complete the search and execute the rest of the commands in the XEQ file. If you respond NO, QUERY will terminate the retrieval command, close the XEQ file, and prompt for another command.

If you enter (CONTROL) Y in response to a value prompt, as in the case of null values and the NODATA option, QUERY will terminate the command, close the XEQ file, and prompt for another command. In some cases, a retrieval command with null values will create an execution loop where the only way to terminate the command is with (CONTROL) Y.

The following is an example of an XEQ file which will create an execution loop. The XEQ file is named FINDACCT and contains:

```
FIND ACCT#=""
REPORT ALL
XEQ FINDACCT,NODATA
```

Examples

Example 1

In this example, an XEQ file called FISET is used to define a particular QUERY session environment. The FISET file contains these records:

```
DEFINE
ORDERS
CLERK
5
CUSTOMER,SALES
PROCX
TERM
```

QUERY executes the DEFINE command and uses the other records as data in response to the DEFINE command prompts. If the NODATA option had been specified, QUERY would prompt you for each DEFINE prompt.

```
>XEQ FISET
DEFINE
DATA-BASE = ORDERS
PASSWORD = CLERK
MODE = 5
DATA-SETS = CUSTOMER,SALES
PROC-FILE  = PROCX
OUTPUT = TERM
OUTPUT = TERM
END OF XEQ FILE
```

Example 2

This example shows how an XEQ file, PRXFIL, can be used to store and execute a pre-planned find-and-report sequence. It also shows the use of null data values and how the FIND CHAIN command can be used to access two data sets. Here, the information from CUSTOMER and SALES is used to prepare a bill.

```
>XEQ PRXFIL,NODATA

FIND CHAIN CUSTOMER.ACCOUNT,SALES.ACCOUNT="" END
WHAT IS THE VALUE OF - ACCOUNT
>>2
2  ENTRIES QUALIFIED
REPORT BILL
     .
     .
     .
```

```
                        BOBO'S MERCANTILE



ACCOUNT #:   2                                  BILLING DATE: 06/19/85


        HARLEY     W       LOND
                  1362 16TH AVE
       SAN FRANCISCO,  CA 94122


PURCHASE DATE                                                PURCHASE
                                                               TOTAL


85/05/05                                                      $418.22


TOTAL DUE                                                     $418.22
```

Figure 3-11. Billing Report From an XEQ File

The XEQ file PRXFIL contains:

```
FIND CHAIN CUSTOMER.ACCOUNT,SALES.ACCOUNT="" END
REPORT BILL
```

Since you want to be prompted for a value when the FIND CHAIN command containing a null data value is called, you use the NODATA option in the XEQ command. Note that if REPORT were called directly from the file, XEQ NODATA would prompt you for report statements instead of reading them from the file. Therefore, the REPORT command is stored as a procedure, BILL, in the current Proc-file, and is initiated from PRXFIL after the FIND CHAIN command.

The procedure BILL contains:

```
REPORT
H1,"BOBO'S MERCANTILE",43,SPACE A4
H2,"ACCOUNT #:",10
H2,ACCOUNT,14
H2,"BILLING DATE:",60,SPACE A2
H2,DATE,70
H3,FIRST-NAME,14
H3,INITIAL,20
H3,LAST-NAME,31
H4,STREET-ADDRESS,31
H5,CITY,20
H5,STATE,25
H5,ZIP,31,SPACE A2
H7,"PURCHASE-DATE",13
H7,"PURCHASE,"70,SPACE B3
H8,"TOTAL",70,SPACE A2
D,PURCH-DATE,8,E1
E1,"XX/XX/XX"
R4,LOAD,QUANTITY
R4,MULT,PRICE
R4,ADD,TAX
D,R4,70,E2
E2,"$$$,$$$.99"
R0,ADD,R4
```

```
TF,"TOTAL DUE",9,SPACE B2,SKIP A
TF,R0,70,E2
S,PURCH-DATE
END
```

A

QUERY/V MESSAGES

The messages in this appendix are listed alphabetically. Messages that begin with a number are listed alphabetically, with the other messages, by the first word after the number. For example, the message: *x* `ENTRIES QUALIFIED` is listed with the messages beginning with the letter E.

MESSAGE	MEANING	ACTION
`A DUMMY DATA SET CAN BE EQUATED TO ONLY ONE DATA SET`	The same dummy data set can only be used in one data set equivalence.	Correct the command and re-enter it.
`A FILE WITH THE GIVEN NAME ALREADY EXISTS`	There is already a file with the same name in the specified group and account.	Either use a different file name or purge the existing file with that name.
`A PARENTHESIS OR BRACKET IS NOT PAIRED IN PATTERN`	A left or right parenthesis or bracket is missing.	Correct the command and re-enter it.
`A SPECIFIED OPTION NOT ALLOWED FOR USER PROCEDURES`	Only the SPACE and SKIP options can be used with user procedures.	Correct the command and re-enter it.
`A TRANSACTION IS IN PROGRESS; TRANSEND MUST BE CALLED BEFORE ANOTHER TRANSBEGIN`	TRANSBEGIN was entered twice without any intervening TRANSEND. Another transaction cannot be started until the preceding transaction has ended.	Enter TRANSEND if you want to end the current transaction.
`ADD OPTION ALREADY SPECIFIED`	The ADD option has been entered twice for the same total statement.	Re-enter the statement with a single ADD option.
`ADD OPTION NOT ALLOWED IN STATEMENT`	A statement other than Total contains an ADD option.	Re-enter the statement without the ADD option.
`ALL DATA SET NAMES ARE VALID NAMES, DATA SET EQUIVALENCES WERE IGNORED`	The specified dummy data set name is an existing data set in one or more open data bases.	Re-enter the command with a valid dummy data set name.
`ALPHA EDIT MASK EXCEEDS MAXIMUM SIZE`	The number of characters in an alphanumeric edit mask exceeds the length of the output device record.	Correct the edit mask length by changing the Edit Statement.

MESSAGE	MEANING	ACTION
ARITHMETIC OVERFLOW [ON R*n*]	The sum of data item values being totaled (using a Total statement ADD option) exceeds 20 digits. A register arithmetic operation caused an underflow/overflow condition on the specified register number *n*.	The REPORT command terminates. Re-enter the command or alter the report procedure and execute it again.
ATTEMPTED DELETION OF CHAIN HEAD	Attempted to delete a master search item value that still exists as a search value in one or more details.	Delete the detail entries, then delete the master entry.
ATTEMPTED MODIFICATION OF A SEARCH OR SORT KEY	The UPDATE REPLACE command has attempted to change the value of a data item defined as a search or sort item.	Delete the entire entry and add it again with a new value for the search or sort item.
ATTEMPTED MODIFICATION OF A READ-ONLY ITEM	The UPDATE REPLACE command has attempted to modify a data item to which you have only read but not write access.	You cannot modify the item unless you supply the appropriate password with the PASSWORD= command.
AUTOMATIC MASTER	The UPDATE ADD command has attempted to add an entry to an automatic master data set. It is not necessary to explicitly enter values for an automatic master. IMAGE will do that for you.	None.
AVERAGE OPTION ALREADY SPECIFIED	The AVERAGE option has been entered twice for the same Total Statement.	Re-enter the statement with a single AVERAGE option.
AVERAGE OPTION NOT ALLOWED IN STATEMENT	A statement other than Total contains an AVERAGE option.	Re-enter the statement without AVERAGE.
BAD DATA BASE REFERENCE	This is an internal QUERY problem.	Contact your HP support representative.
BAD DATA BASE NUMBER	This is an internal QUERY problem.	Contact your HP support representative.
BAD DATA SET OR DATA ITEM REFERENCE	This is an internal QUERY problem.	Contact your HP support representative.

MESSAGE	MEANING	ACTION
BAD ITEM REFERENCE OR BAD LIST (SUB-SYSTEM ERROR)	This is an internal QUERY problem.	Contact your HP support representative.
BAD MODE	This is an internal QUERY problem.	Contact your HP support representative.
BAD PASSWORD	Password not defined for data base. You cannot access any data items or data sets.	Re-enter the password.
BEGINNING OF CHAIN	This is an internal QUERY problem.	Contact your HP support representative.
BEGINNING-OF-FILE	This is an internal QUERY problem.	Contact your HP support representative.
BROKEN CHAIN POINTERS	You have read only access and are sharing the data base with a user who is making structural changes to it.	Re-enter the command.
BUFFER OVERFLOW	The total length of all data items to be modified in the data set exceeds 2048 words and has overflowed the buffer used to hold them. If UPDATE REPLACE, the command terminates.	Use the FORM command to see the exact length of the data items and re-enter the values.
CAN ONLY REPORT FROM DATA BASE USED IN FIND OR SUBSET COMMAND	A data base other than the one specified in the FIND or SUBSET command was specified in your report.	Correct the data base name and re-enter the command.
CAN ONLY RETRIEVE FROM ONE DATA BASE AT A TIME FOLLOWING A FIND COMMAND	A SUBSET on a single data set select file may only refer to one data base.	Correct the data base name and re-enter the command.
CAN ONLY SPECIFY A DATA BASE NAME FOR THE FIRST DATA ITEM OR DATA SET NAMED IN THE COMMAND	The LIST command has qualified a data item other than the first data item with a data base name.	Correct the data base name and re-enter the command.
CANNOT COMPLETE JOIN - MAXIMUM NUMBER OF TEMPORARY FILES IS EXCEEDED	QUERY uses a maximum of 244 temporary files, each 2048 records long. In the process of joining the data sets, this limit has been reached.	Use stricter selection criteria in the MULTIFIND command.

MESSAGE	MEANING	ACTION
CANNOT DELETE ENTIRE COMMAND	You cannot completely delete a command from the command history buffer.	Use the R subcommand if you want to change the entire command.
CANNOT FIND SEARCH STRING IN COMMAND HISTORY BUFFER	The string in the REDO command does not exist in the command history buffer.	Re-enter the command with a valid string.
CANNOT USE SUBSET COMMAND AFTER A FIND CHAIN COMMAND	The SUBSET command cannot be used on a FIND CHAIN select file.	Use JOIN, MULTIFIND, and SUBSET to access the same data.
CHARACTER TYPE DATA ITEMS MAY ONLY BE EQUATED WITH CHARACTER TYPE DATA ITEMS	If one data item named in a data item equivalence is of character data type, the other data item named in the equivalence must also be of character type.	Use the FORM command to check the type of data items.
COMBINED LENGTH OF DATA ENTRIES FORMING A COMPOUND ENTRY CANNOT EXCEED 2047 WORDS	The select file may not be saved since the length of the compound data entry is too large.	None
COMMAND STOPPED AT LIMIT, AS SPECIFIED	Command has retrieved the number of entries specified with #LIMIT parameter.	None
COMMAND TABLE OVERFLOW	LIST: Contains more than 10 logical relationships.	Re-enter the command with fewer logical relationships.
	REPORT: Contains more than 400 statements.	Re-enter report with fewer statements.
	RETRIEVAL: Contains more than 50 logical relationships.	Re-enter command with fewer logical relationships.
x COMPOUND ENTRIES QUALIFIED	*x* is the number of compound entries retrieved by MULTIFIND or SUBSET.	None

MESSAGE	MEANING	ACTION
CONSTANT LITERAL TABLE OVERFLOW	LIST: Number of data item values appearing in the entire LIST command has exceeded the capacity of the table used to hold them. REPORT: Number of literal character strings appearing in all the header, detail, group, and total statements have exceeded the capacity of the table used to hold them.	Re-enter the command. Revise the report and re-enter the command. The maximum number of characters for all literals combined is about 1536 bytes minus twice the number of literals in the report.
CONTROL BREAK INCONSISTENCY	The level numbers of total or group statements do not match properly with the sort statement level numbers.	Revise the report and re-enter the command.
<CONTROL Y>	You have entered (CONTROL) Y.	
COUNT OPTION ALREADY SPECIFIED	The COUNT option has been entered twice for the same total statement.	Re-enter the statement with a single COUNT option.
COUNT OPTION NOT ALLOWED IN STATEMENT	Statement other than total contains a COUNT option.	Re-enter the statement without COUNT.
DATA BASE *data base name* HAS BEEN CLOSED, COMMAND CANNOT BE CONTINUED	The data base that your command refers to has been closed.	Either enter another command that does not access that data base, or use a DEFINE, DATA BASE=, or MULTIDB command to open that data base.
DATA BASE BUSY, DO YOU WISH TO WAIT (YES OR NO)?	The data base was opened with access modes 1 and 5 and other users are currently accessing the data base.	Reply NO to ignore current command. Reply YES to be placed in a wait queue until all other users in the queue have accessed the data base.
DATA BASE CANNOT BE OPENED BECAUSE IT IS DAMAGED	Some of the data in the specified data base is not accurate because the data base has been damaged.	Notify your data base administrator.

MESSAGE	MEANING	ACTION
DATA BASE CANNOT BE OPENED (CHECK !!data base name!! OR SECURITY)	The data base name is invalid, or the account or group specified does not exist, or you do not have access to that data because of MPE security.	Re-enter the command with a valid data base name or change your password with the PASSWORD= command.
DATA BASE CANNOT BE OPENED (DBOPEN DID NOT SUCCEED)	The data base cannot be opened because an error occurred.	If this message persists, notify your data base administrator.
DATA BASE IN USE	The data base is currently being accessed and cannot be opened with an exclusive access mode.	Change the mode or try later.
DATA BASE IS LOCKED BY ANOTHER PROCESS	The data base cannot be accessed because another user has the data base, data sets or entries locked.	Try accessing data base at a later time.
DATA BASE NOT DECLARED	The command has been entered prior to declaring the data base(s).	Use DEFINE, DATA-BASE=, or MULTIDB to declare the data base(s).
DATA BASE OPEN EXCLUSIVELY	The data base is currently being accessed by another user with exclusive access mode.	Wait until the other user closes the data base and try again.
DATA BASE OPEN IN ANOTHER MODE	The data base is currently being accessed in an access mode which is incompatible with the one specified.	Try another mode or try again later. (Refer to Section 1 for compatible modes.)
DATA BASE REQUIRES CREATION	The data base root file exists, but files containing data sets do not.	Run the IMAGE program DBUTIL in CREATE mode to create the data base.
DATA ITEM DOES NOT BELONG TO SPECIFIED DATA SET	The data item does not belong to the data set named in the command.	Re-enter the command with the correct data set name.
DATA ITEM LIST TOO LONG	Internal QUERY error.	Contact your HP support representative.

MESSAGE	MEANING	ACTION
DATA ITEM NOT RETRIEVED *data item name*	The command references a data item that was not previously selected by a retrieval command.	Retrieve entries that contain the data item and re-enter the command.
DATA ITEM USED IN MATCH MUST BE TYPE "U" OR "X"	The data item specified for a pattern match must be of character type.	Use the FORM command to check the type of data items.
DATA ITEM VALUE TOO LONG	The data item value entered exceeds 500 characters or the maximum allowable length for the data item type defined.	Use the FORM command to verify the length of the data item and re-enter with a smaller value.
DATA OF TYPE K CANNOT BE A NEGATIVE VALUE	Data items of type K must be positive.	Re-enter a positive value. See Table 1-1 for the legal value range for type K.
DATA SET BUSY - TRY AGAIN	The data set is being accessed by other users.	Try again later.
DATA SET FULL	No more entries can be added to the data set.	Consult with your data base administrator.
DATA SET NOT WRITEABLE	The command attempted to modify a data item in a data set which you are only allowed to read but not write.	You must supply an appropriate password or enter a different command.
DBB CONTROL BLOCK FULL	Internal IMAGE problem.	Refer to the *IMAGE Reference Manual* for an explanation of the problem.
DBINFO MODE 901 FAILED. CHECK DATA BASE LANGUAGE ATTRIBUTE AND IMAGE VERSION.	The version of IMAGE on your system does not have NLS features.	Refer to the *IMAGE Reference Manual* for an explanation of this problem.
DBG CONTROL BLOCK FULL	Internal IMAGE problem.	This is a warning. The user may want to update IMAGE to a version that supports NLS.

MESSAGE	MEANING	ACTION
DBU CONTROL BLOCK FULL	The IMAGE user control block is full.	Close and reopen the data base you are accessing either by exiting and re-entering QUERY or by using the CLOSE command. If the problem persists, refer to the *IMAGE Reference Manual* for an explanation of the problem.
DELETE ALL RETRIEVED ENTRIES (YES OR NO)?	QUERY requests confirmation of the UPDATE DELETE command just entered.	Reply YES to delete all entries selected by the most recent retrieval command. Reply NO to abort the command.
DIRECTED BEGINNING-OF-FILE	Internal QUERY problem.	Contact your HP support representative.
DIRECTED END-OF-FILE	Internal QUERY problem.	Contact your HP support representative.
DIRECTORY OVERFLOW, PROCEDURE REJECTED	The current Proc-file directory is full and cannot accept more procedures.	Delete existing unused procedures in the current Proc-file or build and declare another one with the PROC-ENTITY = command.
DUPLICATE EDIT STATEMENTS	More than one edit statement with the same number has been entered.	The second edit statement is ignored unless the REPORT is a procedure. If so, RETURN to prompt. Alter the procedure.
DUPLICATE ITEM IGNORED *item name*	The same data item has appeared more than once in the command. Only first occurrence is used.	None
DUPLICATE ITEM NAME	Two or more data sets named in the JOIN command contain the same data item name and one or both of the following: (1) The duplicate items are not named in the data item equivalence. (2) The item is compound.	None

MESSAGE	MEANING	ACTION
DUPLICATE PROCEDURE NAME	The procedure name already exists in the current Proc-file.	Re-enter the command with a different procedure name.
DUPLICATE SEARCH ITEM VALUE	You have entered a search value that already exists in the master data set. The value you tried to add is ignored.	None.
DUPLICATE SORT DATA ITEM NAME	More than one sort statement contains the same data item name, qualified with the same data base name and subscript.	Revise the report and re-enter the command.
EDIT MASK ERROR	A numeric edit mask contains characters in the wrong order.	Correct the mask and re-enter the edit statement.
EDIT OPTION ALREADY SPECIFIED	The statement contains more than one edit statement reference.	Revise the statement and re-enter it.
EDIT OPTION ERROR	The edit statement reference is not between E0 and E9 inclusive.	Revise the statement and re-enter it.
EDIT OPTION NOT ALLOWED IN STATEMENT	The edit statement is referenced in the displayed statement which contains a literal as a print element.	Revise the report and re-enter the command.
EITHER NO JOIN COMMAND WAS ISSUED OR THE MOST RECENT JOIN COMMAND WAS INVALID	The JOIN command was not entered or not entered correctly.	Enter a valid JOIN command.
END OF CHAIN	The end of a detail chain has been prematurely encountered.	Consult with data base administrator. Problem may be an internal QUERY problem or a problem with the data base.
END OF FILE	The end of the data set file has been prematurely encountered.	Consult with data base administrator. Problem may be an internal QUERY problem or a problem with the data base.

MESSAGE	MEANING	ACTION
END-OF-FILE ON COMMAND INPUT FILE	QUERY has encountered the end of the input file before finding an EXIT command. In batch, the end of file could be caused by a :EOD, :EOJ, :JOB or :DATA record.	Check input file and add an EXIT command.
END-OF-FILE ON REPORT FILE	There is not enough room left in the file to which the report is being written.	Notify the system operator.
END OF XEQ FILE	The End-Of-File was encountered in the XEQ file.	None
END PRINT POSITION ERROR	The end print position specified contains an illegal digit, is equal to zero, or is greater than the record size of the output device.	Revise the statement and re-enter it.
END PROCEDURE DATA	The line number referenced in an insert, replace, or delete statement is greater than the final statement number of the procedure being altered.	Re-enter the command with the correct statement number.
x ENTRIES HAVE QUALIFIED, DO YOU WANT TO CONTINUE SEARCHING?	(CONTROL) Y was entered during the retrieval. *x* is the number of entries found so far.	Reply NO if you want to terminate the command. The number of entries that have qualified so far will be preserved. Reply YES if you want to continue.
x ENTRIES NOT DELETED BECAUSE CORRESPONDING DETAIL ENTRIES EXIST	An attempt has been made to delete a master search item value that still exists as a detail search item value in one or more linked detail data sets. *x* is the number of entries not deleted.	If desired, delete the detail entries first and then the master entry.
x ENTRIES QUALIFIED	*x* is the number of entries selected by the command.	None
EOD ON COMMAND INPUT FILE	An :EOD, :JOB, :EOJ, or :DATA record has been encountered.	None

MESSAGE	MEANING	ACTION
ERROR IN COMPILE-MATCH	An internal error occurred while trying to process the pattern specified with MATCHING. There may be too many characters in the pattern.	Try entering a shorter pattern.
ERROR IN SAVE PROCEDURE INVALID ITEM BOUNDARIES - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE INVALID ITEM LENGTH - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE INVALID ITEM OFFSET - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE INVALID ITEM TYPE SPECIFIED - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE ITEM BOUNDARY TOO LONG FOR RECORD - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE ITEM NAME BEGINS WITH NON-ALPHABETIC ITEM - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE ITEM NAME CONTAINS A BLANK - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.

MESSAGE	MEANING	ACTION
ERROR IN SAVE PROCEDURE ITEM NAME CONTAINS INVALID CHARACTER - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE ITEM NAME IS ALL BLANKS - ITEM # *n*	Internal QUERY Problem. Item # n is the number of the item in error.	Contact your HP support representative.
ERROR IN SAVE PROCEDURE NUMBER OF ITEMS MUST BE GREATER THAN ZERO	Internal QUERY Problem.	Contact your HP support representative.
EXPECTED A " , "	A comma is missing from the command or statement.	Correct the command or statement and re-enter it.
EXPECTED A " . "	A period is missing from the command or statement.	Correct the command or statement and re-enter it.
EXPECTED " = "	An equal sign is missing from the command.	Correct the command or statement and re-enter it.
EXPECTED A ";"	A semicolon is missing from the command or statement.	Correct the command or statement and re-enter it.
EXPECTED A CHARACTER IN PATTERN	A pattern must have one or more characters.	Correct the command or statement and re-enter it.
EXPECTED A CONNECTOR OR "END"	The command is missing a logical connector (AND or OR) or a terminating END, or an ending quote is missing from a data item value.	Correct the command or statement and re-enter it.
EXPECTED A DATA ITEM NAME	A data item name is missing from the command.	Re-enter the command with a data item name.
EXPECTED A DATA SET NAME	A data set name is missing from the command.	Re-enter the command with a data set name.
EXPECTED A DUMMY DATA SET NAME	A dummy data set name is missing from the command.	Re-enter the command with a dummy data set name.
EXPECTED A LANGUAGE NUMBER OR NAME	The LANGUAGE=command only accepts the name of a language or the number associated with that name.	Enter HELP LANGUAGE for a complete explanation of the command and then re-enter it.

MESSAGE	MEANING	ACTION
EXPECTED A LITERAL VALUE	The value in quotes is missing or the beginning or ending quote is missing.	If prompted with >>, re-enter the data item name and value. Otherwise, correct the command and re-enter it.
EXPECTED A LEFT PARENTHESIS	The left parenthesis which specifies a sub-item index is missing from the command.	Correct the command and re-enter it.
EXPECTED A RIGHT PARENTHESIS	The right parenthesis which specifies a sub-item index is missing from the command.	Correct the command and re-enter it.
EXPECTED EITHER A COMMA OR A SEMICOLON	The JOIN command expected a comma or semicolon.	Correct the command and re-enter it.
EXPECTED "TO"	A TO is missing from the data item equivalences.	Correct the statement and re-enter it.
FATAL ERROR: QUERY TERMINATED	An unrecoverable error has occurred.	Notify your data base administrator.
FCLOSE FAILURE *x, y*	The MPE intrinsic FCLOSE failed. (IMAGE error -2). *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
FILE CLOSE ERROR *code*	A file error occurred while MPE was closing a file. *Code* is an MPE file system error number.	Notify your data base administrator.
FILE DOES NOT EXIST	The specified file does not exist.	Re-enter the command with a valid file name.
FILE DOES NOT EXIST, BEING CREATED	The procedure file does not exist in the log-on group. The file is created by QUERY and saved using *file name* as the formal designator. File size is 126 records.	None
FILE NOT TYPE ASCII	The referenced file is not an ASCII file.	Re-enter the command using the name of an ASCII file.
FILE OPEN ERROR BAD RECSIZE	The ASCII file declared as the Proc-file does not have a record size of 256 bytes.	Declare a different Proc-file with the PROC-ENTITY = command.

MESSAGE	MEANING	ACTION
FILE OPEN ERROR *code*	A file error occurred while opening a file. *Code* is an MPE file system error number.	Contact your HP support representative.
FILE READ ERROR *code*	A physical error occurred during Proc-file read operation. Code is an MPE file system error number.	Contact your HP support representative.
FILE SYSTEM ERROR OCCURRED IN SDBUILD INTRINSIC - *code*	A file error occurred while QUERY was saving the select file. Code is an MPE file system error number.	Contact your HP support representative.
FILE WRITE ERROR *code*	A physical error occurred during file write operation. Code is an MPE file system error number.	Contact your HP support representative.
'FIND' EXPECTED	The referenced procedure does not contain a FIND command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
FLOCK FAILURE *x, y*	The MPE intrinsic FLOCK failed. (IMAGE error -7) *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
FOPEN FAILURE *x, y*	The MPE intrinsic FOPEN failed. (IMAGE error -1) *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
FREADDIR FAILURE *x, y*	The MPE intrinsic FOPEN failed. (IMAGE error -3). *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
FREADLABEL FAILURE *x, y*	The MPE intrinsic FOPEN failed. (IMAGE eror-4. *x* is the data set number, *y* is the file system error number.	
FULL CHAIN FOR ITEM *item name* IN MASTER *set name*	Command attempted to add a detail entry to a chain which has reached the maximum allowable number of entries. Command ignored. *Item name* is the search item in the detail data set and *set name* is the master data set name.	Consult with your data base administrator.

MESSAGE	MEANING	ACTION
`FULL MASTER FOR` *item name* `IN MASTER` *set name*	An attempt has been made to add a detail data entry with a search item value that does not match any existing search item value in the corresponding automatic master *set name* and a new master entry cannot be created because the automatic master data set is full. *Item name* is the detail data set search item name.	Consult with your data base administrator.
`FWRITEDIR FAILURE` *x, y*	The MPE intrinsic FOPEN failed. (IMAGE error -5). *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
`FWRITELABEL FAILURE` *x, y*	The MPE intrinsic FOPEN failed. (IMAGE error -6). *x* is the data set number, *y* is the file system error number.	Notify your data base administrator.
`HEADERS OVERFLOW A PAGE`	The Header Statements print more header lines of information (counting line skipping) than can fit on one output page.	Revise the Header Statements and re-enter the command.
`ILLEGAL ACCESS`	You do not have write access to the data base because your access mode does not allow it.	Refer to Section 1 for a description of access modes and capabilities associated with each.
`ILLEGAL ASCENDING/DESCENDING CODE`	An ASC or DES sort parameter is missing or incorrectly spelled.	Revise the sort statement and re-enter it.

MESSAGE	MEANING	ACTION
`ILLEGAL DATA ITEM NAME` *item name*	The data item named *item name* does not belong to the data base currently being accessed, is not accessible to you based upon your password, or is not a member of the data set specified in the fully-qualified data item name.	Use the MULTIDB command to open the correct data base, or change your password with the PASSWORD= command, or use the FORM command to determine the correct data item.
`ILLEGAL DATA SET NAME` *set name*	The data set called *set name* does not belong to the data base currently being accessed or you cannot access the data set based on your password.	Re-enter the command with a valid data set name (use the FORM command to determine the valid set names) or change your password with the PASSWORD= command.
`ILLEGAL DATA TYPE`*item name*	The data type specified in a register statement is defined as U or X or a register number is outside the range 0-29.	Revise the register statement and re-enter it.
`ILLEGAL DUMMY DATA SET NAME`	A dummy data set name cannot be longer than 16 characters, and the first character must be alphabetic.	Correct the command and re-enter it.
`ILLEGAL ITEM LENGTH` *x*`, FOR ITEM` *item name*	The referenced data item is not the proper size and type to be processed by QUERY.	Refer to Section 1 for a discussion of data types supported by QUERY.
`ILLEGAL ITEM LENGTH` *x*`, ITEM` *item name* `IGNORED`	The data item being referenced is not the proper size and type to be processed by QUERY. If the data item is a search or sort item, the command terminates. *x* is the data item type.	Refer to Section 1 for a discussion of data types supported by QUERY.
`ILLEGAL ITEM LENGTH` *x* `FOR SEARCH KEY` *item name*	Data item referenced is of a type not supported by QUERY.	Refer to Section 1 for a discussion of data types supported by QUERY.

MESSAGE	MEANING	ACTION
ILLEGAL NAME	ALTER, CREATE, DESTROY, DISPLAY, RENAME: The referenced procedure name exceeds eight characters, contains illegal characters, or consists of a reserved word.	Refer to the CREATE command for naming rules or use DISPLAY LIST to determine the correct procedure name. Re-enter command with correct name.
	DATA-BASE=: Data base name referenced contains an illegal character or more than 24 characters.	Re-enter the command with the correct name.
	OUTPUT=: Name following OUTPUT= is not LP or TERM.	Re-enter command with LP or TERM.
	PROC-ENTITY =: Proc-file name contains an illegal character.	Re-enter command with valid name.
	REPORT: Procedure name exceeds eight characters or does not start with an alphabetic character.	Re-enter command with correct name. Refer to the *MPE Commands Reference Manual* for valid file names.
	XEQ: XEQ file name begins with a non-alphabetic character or is too long.	
ILLEGAL PROCEDURE NAME	The specified procedure name contains non-alphanumeric characters.	Re-enter procedure name without non-alphanumeric characters.
ILLEGAL REGISTER OPERATOR	The operator specified is not one of the valid register operators.	Refer to the Register Statement under the REPORT command for valid operators. Revise the statement.
ILLEGAL SOURCE DIGIT IN CONVERSION CVAD - REPLACED WITH ZERO	Routine which converts ASCII numeric strings to packed decimal format has detected an illegal digit. Digit is converted to zero and processing continues.	None
ILLEGAL SELECTION CRITERIA	The retrieval command contains invalid selection criteria. The syntax may be incorrect.	Refer to Section 3 for the correct syntax of the command. Re-enter the command with a legal selection criteria.
ILLEGAL UPDATE TYPE	The character string following UPDATE is not ADD, REPLACE, or DELETE.	Correct the command and re-enter it.

MESSAGE	MEANING	ACTION
INCONSISTENCY BETWEEN OPTION AND EDIT STATEMENTS	The statement contains an edit statement label not used to define an edit statement.	Revise the statement and re-enter command.
IN ORDER TO SAVE A COMPOUND SELECT FILE, THE NUMBER OF DATA ITEMS THE COMPOUND ENTRY POSSESSES MUST BE LESS THAN 511.	The select file may not be saved since there are too many data items in the compound entry.	None
INPUT ERROR	ALTER: No line number in an insert, delete, or replace statement (for example, /R,).	Re-enter the statement correctly.
	Beginning line number greater than ending statement number (for example, /R,5,3).	
	Beginning line number referenced in statement less than ending line number of previous statement (for example, /R,2,4/D,3)	
	Alter statement does not start with /I,/R,/D, or /E.	
	Illegal character entered.	
	/R or /I entered with no additional lines (for example, /R,4/D,6).	Shorten line if possible and re-enter it.
	CREATE: Input line contains an illegal character or input line exceeds 250 characters.	Re-enter the command
	DISPLAY: The beginning line number is less than 1 (for example, DISPLAY FINDACT, 0,3).	
	The beginning line number is greater than the ending line number (for example, DISPLAY FINDACT, 4,2)	

MESSAGE	MEANING	ACTION
`INPUT TOO LONG`	ALTER: Combined length of continued lines (connected by &) exceeds 250 characters.	Shorten the statement and re-enter the command.
	GENERAL: Command exceeds 698 characters.	Re-enter the command with fewer characters if possible.
	FIND, MULTIFIND, SUBSET, UPDATE: Data item value is longer than the maximum length for the data item.	Re-enter the value.
		None
`INSUFFICIENT VIRTUAL MEMORY`	QUERY cannot obtain necessary system disc space to create the tables and work space required to access the data base.	Try again later.
`INTERNAL QUERY NLS PROBLEM`	While initializing language-dependent information, the NLS subsystem encountered an error from which it could not recover.	Contact your HP support representative.
`INVALID COMMAND`	QUERY does not recognize the command or OUTPUT=LP has been entered twice without an intervening OUTPUT= TERM. The command is ignored the second time it is entered.	Refer to the command description for the proper format and re-enter the command.
`INVALID CHARACTERS AFTER DELETE COMMAND`	The D subcommand can only be followed by other D's or by the insert (I) subcommand.	Enter a valid editing subcommand.
`INVALID CHARACTERS AFTER EDIT COMMAND`	A subcommand of the REDO command is followed by invalid characters.	Refer to the REDO command in Section 3 for information on the editing subcommands.
`INVALID CONNECTOR OR TERMINATOR` *xxxx*	QUERY expected a logical connector (AND or OR) or END. *xxxx* is the offending character string appearing in place of the expected string.	Correct the command and re-enter it.
`INVALID DATA BASE NAME`	The specified data base does not exist.	Re-enter the command with a valid data base name.
`INVALID DATA SET NAME` *set name*	The data set, *set name*, does not belong to the specified data base.	Re-enter the command with a valid set name (use the FORM command to determine the valid set names).

MESSAGE	MEANING	ACTION
INVALID EDIT COMMAND	An invalid REDO editing subcommand was entered.	Refer to the EDIT command in Section 3 for information on the valid subcommands.
INVALID FILE NAME GIVEN	The file name specified must be a legal MPE file name.	Refer to *Using the HP 3000* for a description of legal file names.
INVALID MODE	The response to the MODE= prompt is not an integer from 1 to 8.	If prompted, enter a valid mode, or re-enter the command with a valid mode.
INVALID NAME FOLLOWS FORM COMMAND	There is an error in one of the parameters for the FORM command.	Use HELP to check the FORM syntax and re-enter the command.
INVALID NUMBER FOR COMMAND HISTORY BUFFER	The specified command number does not exist in the command history buffer.	Use LISTREDO to list the commands, then re-enter the command with a valid command number.
INVALID PACKED DECIMAL DIGIT	The data base contains invalid numeric data.	Consult with your data base administrator.
INVALID PASSWORD	The password contains more than eight characters.	If prompted, re-enter the password. Otherwise, re-enter the command with a valid password.
INVALID RANGE FOR COMMAND HISTORY BUFFER	The specified range does not exist in the command history buffer.	Use LISTREDO to list all the commands, then re-enter the command with a valid range of command numbers.
INVALID RELATIONAL OPERATOR	An invalid relational operator has been entered.	Refer to Table 3-1, then re-enter the command with a correct operator.
INVALID SUB-ITEM INDEX	The sub-item index must be a valid integer.	Correct the sub-item index and re-enter the statement.
INVALID ZONED DIGIT	The data base contains invalid numeric data.	Consult with your data base administrator.
INVALID # VALUES FOR RELATIONAL OPERATOR	Multiple data item values only follow "equal" or "not equal" relational operators.	Re-enter the command changing either the relational operator or the values.
JOINING A DATA SET TO ITSELF IS ONLY ALLOWED THROUGH THE USE OF DATA SET EQUIVALENCES	A data set cannot be named twice in the same data item equivalence.	Use a data set equivalence to join a set to itself. Refer to JOIN for a description of dummy data sets.
LANGUAGE INVALID. NATIVE-3000 USED	Language specified not configured. The default, NATIVE-3000 was used.	Ask your system manager what languages are available on your system.

MESSAGE	MEANING	ACTION
LANGUAGE NOT CONFIGURED ON THIS SYSTEM. NATIVE-3000 USED	Languages are configured on each system. Language specified is not available on your system. The default language is NATIVE-3000.	Ask your system manager what languages are available on your system.
LEVEL ERROR	The Level number appearing in a sort, group, or total statement is not valid for the statement.	Revise the necessary statement and re-enter it.
LIMIT MUST BE POSITIVE	The # LIMIT parameter must be greater than or equal to zero.	Correct the command and re-enter it.
LITERAL EXPECTED	A literal is missing from the TRANSMEMO command.	Correct the command and re-enter it.
LITERAL TOO LARGE	The literal character string exceeds the record size of the output device.	Revise the statement containing the character string and re-enter it.
LINE WOULD EXCEED MAXIMUM OF 70 CHARACTERS. BREAK LINE FIRST	The current edit subcommand would cause the current line to be too long.	First use the B or S subcommand to split the line. Then re-enter the current subcommand.
LOGICAL CONNECTIONS ARE INCOMPLETE; COMPOUND DATA SET CANNOT BE GENERATED	It must be possible to trace the data item equivalence from one data set to every other data set named in the JOIN command.	Refer to the JOIN command for more information. Correct the command and re-enter it.
LOGGING ENABLED AND NO LOG PROCESS RUNNING	Logging may have been enabled in DBUTIL for the specified data base and no MPE log process is running, or there may not be enough disc space.	Consult with your data base administrator or refer to the *IMAGE Reference Manual* for more information on logging.
LOGGING NOT ENABLED FOR THIS USER	You have opened the data base for read access only and there is no need to log transactions.	Refer to the *IMAGE Reference Manual* for more information on logging.
MATCH NOT VALID WHEN LANGUAGE <> NATIVE-3000	QUERY can only allow the matching option for NATIVE-3000.	If possible, change the language to NATIVE-3000 for the match.
MAXIMUM NUMBER OF DATA BASES (10) ALREADY OPEN	No more data bases may be opened since the maximum number of data bases that can be open at one time is (10).	One of the data bases opened with the MULTIDB command can be closed with the CLOSE command.

MESSAGE	MEANING	ACTION
MISSING SEARCH KEY VALUE FOR *item name* IN MASTER *set name*	Command tried to add an entry to a detail data set linked to the specified master data set without a corresponding entry in the master data set for the search item value. *Item name* is the detail data set search item and *set name* is the master data set name.	Add an entry with the same search item value to the master data set and then re-enter this command.
MISSING SEARCH OR SORT ITEM	Internal QUERY problem.	Contact your HP support representative.
MORE THAN 5 FIELDS ARE BEING TOTALED	The total statements referenced more than five different data items.	Revise the necessary total statements and re-enter the command.
MORE THAN 10 PATTERNS IN ONE RETRIEVAL COMMAND	There may be no more than ten patter matches in one retrieval command.	Correct and command and re-enter it.
'MULTIFIND' EXPECTED ||The referenced procedure does not contain a MULTIFIND command. ||Re-enter the command with the correct procedure name. =MUST OPEN THE FIRST DATA BASE WITH EITHER THE DEFINE COMMAND OR THE DATA-BASE= COMMAND	The MULTIDB command may only be used after the primary data base is open.	Use the DEFINE or DATA-BASE= command to open the primary data base, then re-enter the command.
MUST SPECIFY BOTH DATA BASE NAME AND DATA SET NAME IN USER DEFINED PROCEDURE	If the data base name is specified in a user procedure, then the data set name must be also, and vice versa.	Specify the missing parameter in your user procedure, recompile and store into the proper SL.
NEW COMMAND EXCEEDS MAXIMUM OF 698 CHARACTERS	The length of the edited command exceeds the QUERY maximum of 698 characters.	Reduce the command length by deleting characters.
NLCOLLATE INTRINSIC INTERNAL ERROR	An unexpected error occurred during a data comparison.	Contact your HP support representative.
NLUTIL INTRINSIC INTERNAL ERROR	The NLS subsystem encountered an error from which it could not recover while attempting to initialize language-dependent information.	Contact your HP support Representative.
NO DATA BASE NAME GIVEN	A data base name is missing from the command.	Re-enter the command with the data base name.

MESSAGE	MEANING	ACTION
NO DATA BASE WITH THE GIVEN NAME IS CURRENTLY OPEN	The specified data base name is invalid or not currently open.	Use DEFINE and MULTIDB to see which data bases are open; correct the command and re-enter.
NO ENTRY	No entries match the selection criteria.	None
NO COMMAND TO EDIT	The command history buffer is empty.	None
NO FILE NAME GIVEN	A file name was expected in the command.	Correct the command and re-enter it.
NO RECORDS HAVE BEEN RETRIEVED YET - SUBSET COMMAND NOT ALLOWED	The SUBSET command was entered before entries were retrieved with a FIND or MULTIFIND command.	First use a FIND or MULTIFIND command then re-enter the SUBSET command.
NO RECORDS TO REPORT	No records have been retrieved yet.	First use FIND or MULTIFIND, then re-enter the REPORT command.
NO RETRIEVAL WAS MADE FROM DATA BASE *data base name*, WHICH WAS NAMED IN A USER PROCEDURE	The data base specified is not the name of a currently opened data base.	Correct the data base name in your user procedure, recompile, and store into the proper SL.
NO RETRIEVAL WAS MADE FROM DATA SET *data base name:* data set name	The FIND, MULTIFIND or SUBSET command did not access this data set.	Enter another FIND, MULTIFIND or SUBSET command accessing this data set, or correct your report statement.
NO SUCH DATA BASE	The referenced data base does not exist or you did not include the account or group name.	Re-enter the command with the correct data base name or append a group or account and try again.
NO TRANSACTION IS IN PROGRESS	TRANSEND was entered before TRANSBEGIN; TRANSBEGIN is needed to start a transaction.	Use TRANSBEGIN if you wish to start a transaction.
NO. OF SORT ITEMS EXCEEDS LIMIT	The number of sort items defined exceeds the maximum of 66.	Revise the report and re-enter the command.

MESSAGE	MEANING	ACTION
NON-NUMERIC DIGIT ENCOUNTERED	The value entered contains an illegal character.	Enter the correct value.
NON-NUMERIC IN REAL VALUE	The value entered for a data item type real contains an illegal character, or the value is larger than the maximum allowed for the data type. (This does not include scientific notation.)	Refer to Section 1 for information on data item types and ranges. Re-enter the command with a correct value.
NOT A FIND PROCEDURE	The referenced procedure does not contain a FIND command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
NOT A JOIN PROCEDURE	The referenced procedure does not contain a JOIN command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
NOT A MULTIFIND PROCEDURE	The referenced procedure does not contain a MULTIFIND command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
NOT A SUBSET PROCEDURE	The referenced procedure does not contain a SUBSET command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
NOT A REPORT PROCEDURE	The referenced procedure does not contain a REPORT command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
NOT A REPORT STATEMENT	The statement is not recognized as a valid report statement.	Revise the statement and re-enter it.
NOT AN UPDATE PROCEDURE	The referenced procedure does not contain an UPDATE command.	Use DISPLAY and DISPLAY LIST to determine correct procedure name. Re-enter the command.
NOT ALLOWED TO USE "MATCHING" WITH FIND CHAIN	MATCHING cannot be used with the FIND CHAIN command.	Use JOIN and MULTIFIND to access the same data.
NOT ENOUGH SECTORS IN QSKIB FILE	More data item values appear in a retrieval command than can be stored in a temporary file known as QSKIB. The QSKIB file can contain approximately 12,000 characters.	Re-enter the command with fewer data item values.

MESSAGE	MEANING	ACTION
NOREPEAT MAY ONLY BE ASSOCIATED WITH UP TO 10 SORT LEVELS AND/OR REGISTERS	The number of registers which use the NOREPEAT option plus the number of sort levels for which the total statement uses NOREPEAT must be less than or equal to 10.	Correct the report.
NOREPEAT OPTION ALLOWED ONLY IN REGISTER OR TOTAL STATEMENTS	NOREPEAT cannot be used as a REPORT option on any statement types other than Register or Total.	Correct the statement and re-enter it.
NUMBER SPECIFIED IS TOO LARGE/SMALL FOR DATA ITEM TYPE *x*	The value entered is too large or two small for data type *x*.	Enter the correct value. Refer to Table 1-1 for value ranges of different data types.
NUMERIC EDIT MASK EXCEEDS 20 CHARACTERS	The numeric edit mask in an edit statement exceeds the maximum allowable 20 characters.	Revise the edit mask and re-enter the statement.
NUMERIC VALUE ERROR *or* NUMERIC VALUE ERROR ITEM IGNORED ||ALTER, DISPLAY: A line number contains too many digits or an illegal character. ||Re-enter the statement or command with the correct line number. ||LIST, UPDATE: A numeric value contains an illegal character or is larger than allowed for the data type. |Re-enter the command with the correct value. =OLD FORMAT ON PROCEDURE FILE	The file specified as the Proc-file does not have a file code of 1070.	Ask the data base administrator for help.
OPTION ERROR	The statement option is either illegal for the statement in which it appears, the option is not recognized as valid, or you have used NOREPEAT in a single data set report.	Revise the statement and re-enter it.
OUTMODED ROOT FILE	The data base root file was created using a different version of the IMAGE software. Current QUERY software is not compatible with this file.	Consult with your data base administrator.

MESSAGE	MEANING	ACTION
PARAMETER ERROR	A syntax or spelling error has occurred.	Revise the statement and re-enter the command.
PASSWORD DOES NOT MATCH ANY DEFINED FOR SPECIFIED DATA BASE. USER CLASS ZERO (0) WAS ASSIGNED	The password was typed incorrectly.	Re-type the password.
PASSWORD ERROR	The password contains more than eight characters or does not allow at least read access to the data base.	If prompted, enter the correct password. Otherwise, re-enter the command with the correct password.
PROCEDURE CANNOT BE DELETED BY ALTER	You have attempted to delete a procedure, line by line, using ALTER. The procedure is not changed and the ALTER command terminates.	To delete a procedure, use the DESTROY command.
PROCEDURE NAME NOT FOUND	The named procedure is not in the directory for the current Proc-file. This message may also appear if the command syntax is incorrect.	Correct the procedure name, declare a different Proc-file, or correct the command syntax.
PROCEDURE NAME TOO LONG	The name of the procedure is more than eight characters.	Re-enter the command with a valid procedure name.
PROC-FILE BUSY	The Proc-file is currently being accessed by another user.	The command using the Proc-file is ignored. Try again later.
PROC-FILE BUSY, DO YOU WISH TO WAIT (YES OR NO)?	The Proc-file is being accessed by another user. In batch mode the default answer is YES.	Reply NO to abort the command. Reply YES to be placed in wait queue until all users ahead of you in queue have accessed the file.
PROC-FILE NOT DECLARED	The command has referenced a procedure before a Proc-file has been declared.	Use the PROC-ENTITY = command to declare a Proc-file.
PROC-FILE OVERFLOW, INPUT TERMINATED	The procedure being entered or altered is too large for the remaining space in the Proc-file. Part of the procedure will be lost.	Either delete unused procedures in the Proc-file or create the procedure in another Proc-file.
PROC-FILE TOO LARGE - MUST BE <= 400 RECORDS	The file being declared as the Proc-file is larger than 400 records.	Re-enter the command with a different file name.

MESSAGE	MEANING	ACTION
PROC-FILE TOO SMALL - MUST BE >= 5 RECORDS	The file being declared as the Proc-file is smaller than five records.	Re-enter the command with a different file name.
'QSLIST' DEVICE NOT AVAILABLE	The device defined as QSLIST is in use by another process.	Wait and try again later or use the terminal.
QUERY IS NOT ALLOWED TO ACCESS THIS DATA BASE	The data base administrator has disallowed all access to this data base.	Use another data base.
QUERY IS NOT ALLOWED TO MODIFY THIS DATA BASE - USE MODES 5-8	The data base administrator has disallowed all write access to this data base.	Use the MODE=, DEFINE, DATA-BASE=, or MULTIDB command to specify a mode greater than or equal to 5.
RANGE IN PATTERN HAS FIRST CHARACTER GREATER THAN SECOND CHARACTER	If a range is specified in pattern matching, the first number must be less than or equal to the second number.	Correct the command and re-enter it.
READ ERROR FROM COMMAND INPUT FILE	A physical read error occurred while reading from the file used for input to QUERY.	Check the input device and try entering the command again. Consult with the system manager.
REAL VALUE ERROR - ITEM IGNORED	Real value contains an illegal character or is not in the proper range for the data item defined.	Refer to Section 1 for data item types and ranges. Re-enter the command with a correct value.
RECORD HAS NOT YET BEEN FOUND	The command has been executed prior to selecting entries with a retrieval command. UPDATE DELETE or UPDATE REPLACE: The command was used after a multiple data set retrieval.	First use a retrieval command and then re-enter the command. Use a FIND, FIND ALL or SUBSET (following a FIND or FIND ALL) and then re-enter the command.
RECORDS = *x*	This message is printed in response to CREATE SPACE. *x* is the number of unused records in the Proc-file.	None
REDO ONLY AVAILABLE DURING INTERACTIVE SESSION	The REDO command cannot be issued from a job file or a command file.	Remove the command.
REPORT CANNOT BE GENERATED DUE TO ERRORS	The REPORT command contains errors and will not generate a report.	Revise the report and re-enter the command.

MESSAGE	MEANING	ACTION
`REPORT INCOMPLETE - ITEMS MISSING`	Some of the data items selected by the retrieval command may have been deleted by some other user prior to executing the REPORT command.	Re-enter the retrieval and REPORT commands.
`REQUESTED ACCESS MODE IS UNAVAILABLE`	The data base is currently being accessed in an access mode which is incompatible with the one specified.	Try another mode, or try again later. (Refer to Section 1 for compatible modes.)
`RETRIEVAL FROM MORE THAN ONE DATA SET`	The LIST or FIND command has specified access to more than one data set or the FIND CHAIN command has specified access to more than one detail data set.	Use the FORM command to determine the data base structure. Re-enter the command.
`SAME LINES HAVE CONFLICTING REPORT OPTIONS` *label*	A statement option has appeared more than once in statements concerning the same report line. *Label* is the statement type and level.	Revise the necessary statements and re-enter the command.
`SCRATCH FILE` { `READ` / `OPEN` / `WRITE` } `ERROR` *code*	A physical error has occurred during QUERY access of a scratch file. The file is not accessible to the user. *Code* is the file system error number.	Contact your HP support representative.
`SECURITY VIOLATION`	MPE file security has been violated (for example, a user with read access attempts to open a file with write access.)	Try another mode or log-on account or ask the data base administrator for help.
`SELECT FILE IS EMPTY - NO RECORDS HAVE BEEN FOUND YET`	The SAVE command was entered before a select file was established with either FIND, SUBSET, or MULTIFIND.	First use either FIND, SUBSET, or MULTIFIND, then re-enter the SAVE command.

MESSAGE	MEANING	ACTION
SETLOCKS COMMAND ALREADY IN EFFECT	SETLOCKS was previously entered without an intervening RELEASE command.	Enter RELEASE to cancel the locks already in effect. If RELEASE is not entered, any new locks will be held along with any locks already in effect.
SETLOCKS COMMAND WAS NOT IN EFFECT	SETLOCKS was not previously entered.	None
SKIP OPTION ALREADY SPECIFIED	The SKIP option has been specified twice in the same statement.	Revise the statement and re-enter the command.
SKIP OPTION ERROR	The SKIP option is not SKIP A *n* or SKIP B *n* where *n* is 1 to 5 inclusive.	Revise the statement and re-enter the command.
SKIP OPTION NOT ALLOWED IN HEADER	The header statement contains a SKIP option.	Revise the statement and re-enter the command.
SORTLIB: *message*	An error has occurred in one of the SORT/3000 procedures operating in behalf of QUERY. *Message* is the SORT/3000 error message.	Refer to the *SORT/3000 Reference Manual* and try the command again.
SORT LEVEL MISSING OR DUPLICATE *label*	The group or total statements reference sort statements which either do not exist or appear twice in the REPORT command. For example, S1 appears twice, or a total statement is labeled T3 when no S3 statement exists. *Label* is the statement type and level.	Revise the necessary statements and re-enter the command.
SPACE OPTION ALREADY SPECIFIED	A SPACE A or SPACE B option appears twice in the same statement.	Revise the necessary statements and re-enter the command.
SPACE OPTION ERROR	A SPACE option is not SPACE A *n* or SPACE B *n*, where *n* is 1 to 5 inclusive.	Revise the necessary statements and re-enter the command.
SPECIFIED FILE CALLS FOR CARRIAGE CONTROL CHARACTERS (CHECK FOR FILE EQUATE)	The file named in the SAVE command cannot have carriage control characters.	Either use another file name or refer to the *MPE Commands Reference Manual* for the FILE and RESET commands.

MESSAGE	MEANING	ACTION
SPECIFIED FILE HAS A NON-SDFILE FILECODE (CHECK FOR FILE EQUATE)	The file named in the SAVE command must have an SDFILE filecode.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS INCOMPATIBLE RECORD SIZE (CHECK FOR FILE EQUATE)	The file named in the SAVE command has an incorrect record size.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS INCOMPATIBLE FILE LIMIT (CHECK FOR FILE EQUATE)	The file named in the SAVE command has an illegal file limit.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS NON-FIXED-LENGTH RECORDS (CHECK FOR FILE EQUATE)	The file named in the SAVE command must have fixed-length records.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS MULTIRECORD ACCESS (CHECK FOR FILE EQUATE)	The file named in the SAVE command cannot have multirecord access.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS NOBUF (CHECK FOR FILE EQUATE)	The file named in the SAVE command cannot have NOBUF.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE HAS NOWAIT I/O (CHECK FOR FILE EQUATE)	The file named in the SAVE command cannot have NOWAIT I/O.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE IS NOT AN ASCII FILE (CHECK FOR FILE EQUATE)	The file named in the SAVE command must be ASCII.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.
SPECIFIED FILE IS NOT ON DISC (CHECK FOR FILE EQUATE)	The file named in the SAVE command must be a disc file.	Correct the file equate (refer to Appendix G for SAVE file specifications) or use another file name.

MESSAGE	MEANING	ACTION
STATUS=%*XXXXXX* P=%*XXXXXX* TRAP=%*XXXXXX*	A hardware trap has occurred which cannot be handled by QUERY. (May be caused by hardware failure.) QUERY terminates. *XXXXXX* is an octal number. STATUS is the status register, P is the P-register and TRAP is the MPE trap number.	Consult with the system manager.
SUB-ITEM INDEX MUST BE 1 OR GREATER	The sub-item index must be an integer greater than 0 and less than or equal to the number of sub-items that the data item possesses.	Correct the sub-item index and re-enter it. Use the FORM command to check the number of sub-items for that data item.
SUB-ITEM INDEX TOO LARGE	The sub-item index specifying which sub-item to use is greater than the number that the data item possesses.	Use FORM to check the number of sub-items for that data item and re-enter the command.
'SUBSET' EXPECTED	The referenced procedure does not contain a SUBSET command.	Use DISPLAY or DISPLAY LIST to determine the correct procedure name. Re-enter the command.
SUBSYSTEM BUFFER TOO SMALL	Internal QUERY problem.	Contact your HP support representative.
TEXT MUST BE NO GREATER THAN 512 CHARACTERS	The memo on the log file cannot exceed 512 characters.	Re-enter the log file memo.
THE DATA BASE *data base name*, NAMED IN THE USER PROCEDURE, IS ILLEGAL.	The specific data base is either an illegal name or is not the name of a currently open data base.	Correct the data base namein your user procedure, recompile it, and store it into the proper SL.
THE DATA ITEM *data item name* DOES NOT BELONG TO ANY DATA SET NAMED IN THE MOST RECENT JOIN COMMAND	The value of this data item cannot be reported since it does not belong to any data set that was accessed by the MULTIFIND command.	Correct the command and re-enter it. The FORM command can be used to determine what items are in each set in the JOIN command. The SHOW JOIN command will show you the current JOIN.
THE DATA SET *data set name*, NAMED IN A USER PROCEDURE, IS ILLEGAL ||The data set specified in the user procedure is not in the specified data base, or is not accessible to you based upon your password.	Correct the data set name in your user procedure, recompile, and store into the proper SL, or change your password.	

MESSAGE	MEANING	ACTION
THE DATA SET *data set name*, NAMED IN A USER PROCEDURE, IS NOT IN DATA BASE *data base name*.	The data set specified in the user procedure does not belong to the data base specified, or is not accessible to you based upon your password.	Correct the data set name in your user procedure, recompile, and store into the proper SL, or change your password.
THE DATA SET *data base name: data set name* WAS NOT NAMED IN THE MOST RECENT JOIN COMMAND	A data set named in the MULTIFIND or SUBSET of a MULTIFIND must have been specified in the most recent JOIN command.	Enter SHOW JOIN to see the most recent JOIN command.
THE SAVE COMMAND MAY NOT BE USED IF THE @ SIGN WAS SPECIFIED IN THE JOIN COMMAND	The JOIN command contained the optional @ parameter.	None
THE @ SIGN IS NOT ALLOWED ON A CYCLE OF DATA SETS	The @ sign is not allowed in a JOIN command if the data item equivalences form a cycle.	Refer to Section 3 (under the JOIN command) for more information. Correct the command and re-enter it.
THE @ SIGN IS NOT ALLOWED ON BOTH SIDES OF THE "TO"	There can only be one @ per data item equivalence.	Correct the command and re-enter it.
THIS COMMAND HAS GONE OVER THE LIMIT OF 260,064 ENTRIES; 260,064 ENTRIES HAVE BEEN KEPT	Only 260,064 entries can be selected as a result of a single FIND or MULTIFIND command.	Re-enter the command with stricter selection criteria or use the entries already selected.
THIS DATA BASE IS ALREADY OPEN FOR THIS JOB OR SESSION	A data base can only be opened once within QUERY.	Use the DEFINE or MULTIDB command to alter any environment specifications for a particular data base.
THIS DATA BASE WAS OPENED WITH THE DEFINE COMMAND	This data base is the primary data base and a data base can only be opened once within QUERY.	To change a data base from being a primary data base to a data base opened with the MULTIDB command, first open a new primary data base with the DEFINE or DATA-BASE= command. Then open the previous primary data base with the MULTIDB command.

MESSAGE	MEANING	ACTION
TOO MANY DATA ITEM EQUIVALENCES WERE SPECIFIED	There can only be up to 53 data item equivalences.	Decrease the number of equivalences and re-enter the command.
TOO MANY DATA SET EQUIVALENCES WERE SPECIFIED; JOIN EXPECTED	There may only be up to 53 total data item and data set equivalences.	Decrease the number of equivalences and re-enter the command.
UNABLE TO ACCESS DATA BASE IN THIS MODE, *m, n*	The root file cannot be opened with the access options required for the specific MODE. *m* is the list of required options and *n* is the list of options granted by MPE file system. (*m, n* are integers.)	Ask the data base administrator for help.
UNABLE TO USE FILE *name* ||The requested new Proc-file *name* is not large enough or is not an empty existing Proc-file. ||Re-enter the command with a different *name*. =UNDO COMMAND NOT ALLOWED NOW	The UNDO command is only allowed following a SUBSET command.	Refer to the SUBSET command for correct usage.
UNEXPECTED ERROR IN REPORT PRINT: REPORT COUNT TABLE FULL	Internal QUERY problem.	Contact your HP support representative.
UNIFYDETAIL MAY NOT BE USED - HEADER AND DETAIL SPACE DOES NOT FIT ON ONE PAGE	The header and detail lines will not all fit on one page, of may not be used for this report.	Correct the report.
UNLOAD PROC FAILURE	The MPE intrinsic UNLOADPROC failed. This error indicates hardware or system software failure.	Notify the system manager.
UPDATE TO MORE THAN ONE DATA SET	The UPDATE DELETE and UPDATE RELACE commands cannot be used on entries selected from more than one data set.	Use the FIND or SUBSET command to select entries to be deleted or replaced.
USE PLUS OR MINUS SIGN	To move backward, you must use a minus sign. To move forward, you must use a plus sign.	Re-enter the REDO editing subcommand.

MESSAGE	MEANING	ACTION
`USER LANGUAGE INVALID`	User language not available. Only NATIVE-3000 is available on your system.	Ask the System Manager to configure the desired language on your system.
`USER LANGUAGE NOT CONFIGURED ON THIS SYSTEM. NATIVE-3000 USED.`	Languages are configured on each computer system. Language specified is not available on your system. The default language is NATIVE-3000.	Ask your system manager what languages are available on your system.
`USING SERIAL READ`	QUERY must read each record of the detail data set without the benefit of a master data set search key.	Wait for QUERY to complete the search, or terminate the command with CONTROL Y.
`USING SORT/MERGE`	Sorting and merging of data sets must be used in the searching of the compound data set being created by the MULTIFIND command.	Wait for QUERY to complete the search, or terminate the command with CONTROL Y.
`WARNING: LOCKS ARE BEING HELD`	SETLOCKS has been entered and there are locks in effect that will remain until RELEASE is entered.	If you do not need to retain these locks, enter a RELEASE command since other users may want to access the locked data set(s).
`WHAT IS THE VALUE OF` *item name*	The command contains a null data item value. You are prompted for the data item value.	Supply the requested data item value.
`WRITE ERROR TO COMMAND LISTFILE`	A physical write error occurred while writing to the standard list device ($STDLIST).	Check the output device and try the current command again. Consult with the system manager.
`@ SIGNS MUST BE PROPAGATED`	@ must propagate along data item equivalences.	Refer to the JOIN command. Correct the command and re-enter.
`$MISSING MAY ONLY BE USED FOR MULTIPLE DATA SET ACCESS`	$MISSING cannot be used as a value for a SUBSET command on a single data set select file or for the FIND CHAIN command.	Correct the command and re-enter it.

B

QUERY/3000 SPECIFICATIONS

LIMITS USED BY QUERY

Table B-1. General

TYPE	MAXIMUM
Data Base Name	6 characters (excluding group and account)
Number of Open Data Bases	10
Password Length	8 characters
Record Size of Output Device	68 words
Size of Data Item Value	500 characters (250 words)
Size of Input Line	698 characters

Table B-2. JOIN Command

TYPE	MAXIMUM
Length of Dummy Data Set Name	16 characters
Number of Data Item Equivalences	52
Number of Data Sets in JOIN	53

Table B-3. LIST Command

TYPE	MAXIMUM
LIST Literals	768 total characters
LIST Logical Connectors	10

Table B-4. PROC-FILE Command

TYPE	MAXIMUM
ALTER Command Input Line	250 characters
Number of Procedures in a Proc-file	Refer to PROC-FILE under "FILES USED BY QUERY"
Proc-file Name	8 characters

Table B-5. REPORT Command

TYPE	MAXIMUM OR RANGE
Alphanumeric Edit Mask Length	136 characters
Detail Levels	1-99
Edit Levels	0-9
Group Levels	1-10
Header Levels	1-9
Line Length	136 characters
Literals	1536 total characters (including quotation marks)
Number of NOREPEAT Items	10 (registers plus total items)
Numeric Edit Mask Length	20 characters
Register Numbers	1-30
REPORT Literals	136 total characters
Sort Item Length	136 characters
Sort Levels	1-9
Sort Statements	66 unnumbered
Total Statement Data Items	5
Total Levels	1-10 and TF
Total Number of REPORT Statements	400

Table B-6. Retrieval Command (FIND, MULTIFIND, SUBSET)

TYPE	MAXIMUM
FIND Chain Values	16
FIND, MULTIFIND, SUBSET Literals	12288 total characters
FIND, MULTIFIND, SUBSET Logical Connectors	50
FIND, MULTIFIND, SUBSET Retrieval	260,064 entries
Number of Patterns used with MATCHING	10

Table B-7. SAVE Command

TYPE	MAXIMUM
SAVE Compound Entry	2047 words in length, 510 data items

Table B-8. TRANSMEMO Command

TYPE	MAXIMUM
TRANSMEMO Text	512 characters

Table B-9. Updating Command (UPDATE ADD and UPDATE REPLACE)

TYPE	MAXIMUM
REPLACE or ADD Entry	2048 words (IMAGE maximum)

FILES USED BY QUERY

Table B-10. Files Equated by the User

FILE NAME	DESCRIPTION
QSIN	The file opened by QUERY for input. Opened as a new file. Can only be redirected on QUERY versions C.00.00 and later.
QSOUT	The file opened by QUERY for output. Opened as a new file. Can only be redirected on QUERY versions C.00.00 and later.
QSLIST	The output device specified when OUT=LP. Refer to the OUTPUT= command for details on equating this to a device other than the line printer.
QSMSGCAT	The QUERY message catalog, located in pub.sys. File is opened fully qualified.

Table B-11. Files Not Equated by the User (Temporary Files)

FILE NAME	DESCRIPTION
QLSREDOF	This is a temporary file which contains the command history buffer used by the LISTREDO and REDO commands.
QREDOF	This is a temporary file which contains the command being edited with the REDO command.
QSCMPSEL	A select file. This is a temporary file which contains the record numbers of the compound data set entries which qualify from a MULTIFIND or SUBSET command.
QSKIB	Holds the literal values specified in the FIND, MULTIFIND or SUBSET selection criteria.
QSOLDSEL	A select file which is created when the SUBSET command is entered. This is the file which is restored when an UNDO command is entered. This is a temporary file which contains the contents of the previous select file. The previous select file contains the record numbers of the entries which qualified from a FIND or MULTIFIND command.
QSSELECT	A select file. This is a temporary file which contains the record numbers of the single data set entries which qualify from a FIND or SUBSET command.

Table B-12. Files Created by the User and/or Specified to be Built by QUERY

FILE NAME	DESCRIPTION
PROC-FILE	Contains the procedures that may be used for executing certain QUERY commands. Opened as an old file. Required File Attributes: file code 1070, ASCII, rec size 128 words file size must be at least 5 records and not more than 400 records. QUERY will build the file with 126 records. Note that most commands, with the exception of large reports, will require only one record.
SAVE FILE	A self-describing (SD) file whose internal format is outlined in Appendix E. An SD file contains the entries which qualified from a FIND, MULTIFIND or SUBSET command. Cannot be built by you. The file must be a new file. You can specify a file equate. Required File Attributes: file code 1084, ASCII, fixed length records, disc file required defaults: NOCCTL, NOMR, BUF, WAIT record size: size in bytes of IMAGE entry file size: number of entries to be saved

Table B-13. Unnamed Scratch Files (Temporary Files)

USED BY	DESCRIPTION
JOIN	Used to hold the most recent JOIN command, if any.
MULTIFIND	Used to create the compound data set specified in the JOIN command. As each data set is joined with the previous ones, the record numbers are held in this file. In this way the file holds "intermediate" compound data sets until the final data set is joined, which results in the final select file - QSCMPSEL. 244 files, each 2048 records long
REPORT	Used for the NOREPEAT option.

C

ACCESSING A REMOTE DATA BASE

If you want to use QUERY to access a data base that resides on a remote HP3000, there are three methods available using Distributed Systems (DS/3000) and/or Network Services (NS/3000). Included in this section is information on both DS and NS. Either of these services can be used in Methods 1 or 2. Refer to the note in Method 3 for restrictions when using a data-base-access file. For complete instructions on operating remote sessions, refer to the *DS/3000 Reference Manual* or the *NS/3000 User/Programmer Reference Manual*.

METHOD 1: QUERY ON REMOTE SYSTEM

Method 1 is the most efficient since it minimizes communications line traffic. You can log on to your local computer, establish a communications link, initiate a remote session, and run QUERY on the remote system. This method is illustrated in Figure C-1. A user whose user name is USERL and whose account is ACCTL logs on to the local system. The user then establishes a communications link with the remote system SYSR, known as the parameter *dsdevice* in DS or *envid* in NS. A REMOTE HELLO command establishes a session on the remote system with user identification MEMBER.PAYROLL. The user runs QUERY on the remote system accessing a data base named DBR.

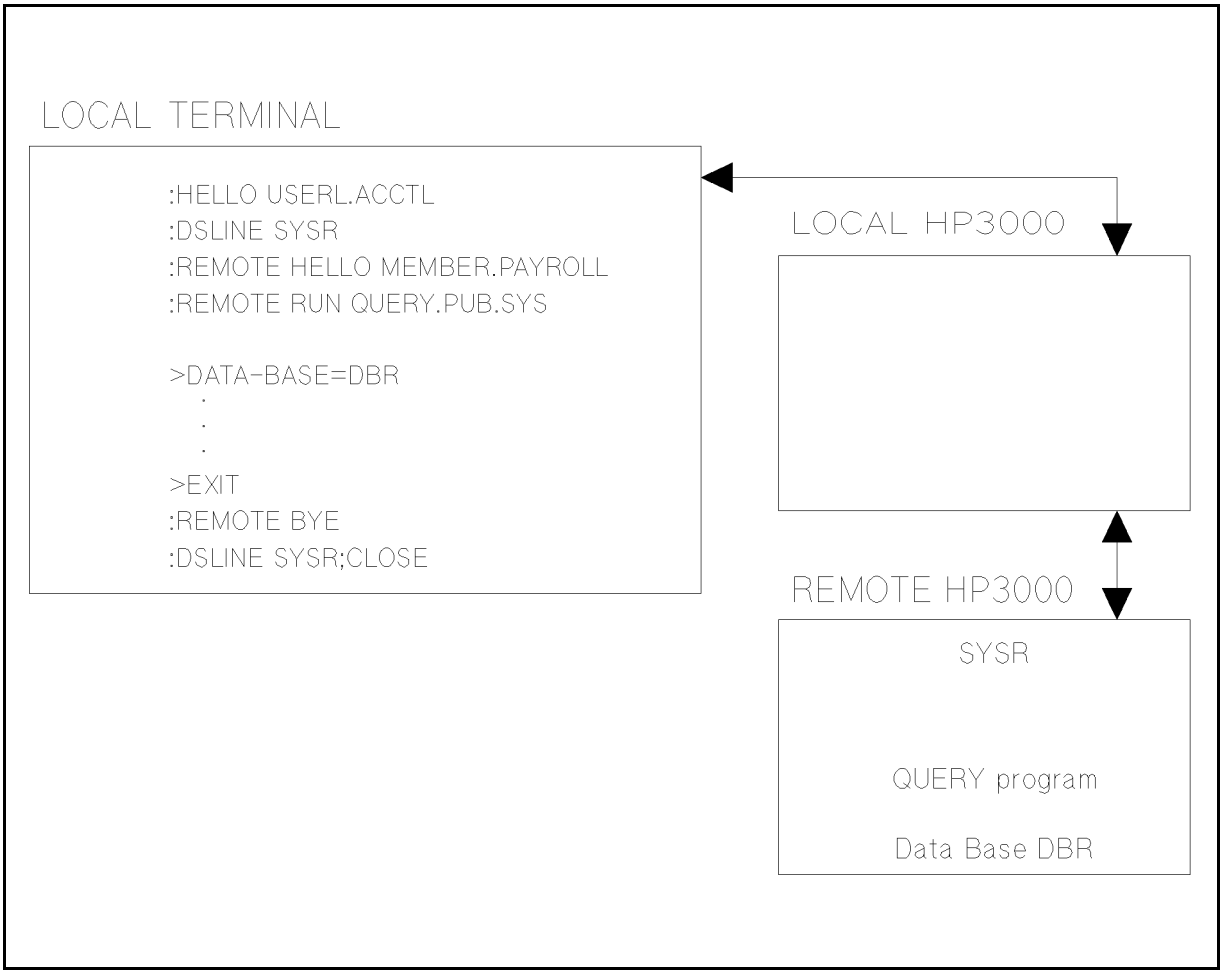


Figure C-1. Using Method 1

METHOD 2: QUERY ON LOCAL SYSTEM

The second method is similar to Method 1 except QUERY is run on the local system. First a :FILE command is used to specify the location of the remote data base. After initiating execution of QUERY, you specify the data base name with the DEFINE or DATA BASE= command. The name of the data base must be the same as the name specified as the formal file designator in the :FILE command.

Figure C-2 illustrates Method 2 using DS. After USERL establishes a communications link and remote session on the remote system, the user enters a file equation to specify that the data base DBR resides on a remote system with dsdevice name SYSR. The device type may be omitted if the default device type is acceptable. The user runs QUERY on the local system and specifies the data base name DBR with the DATA-BASE= command. Note: The data base name is the formal file designator in the :FILE command.

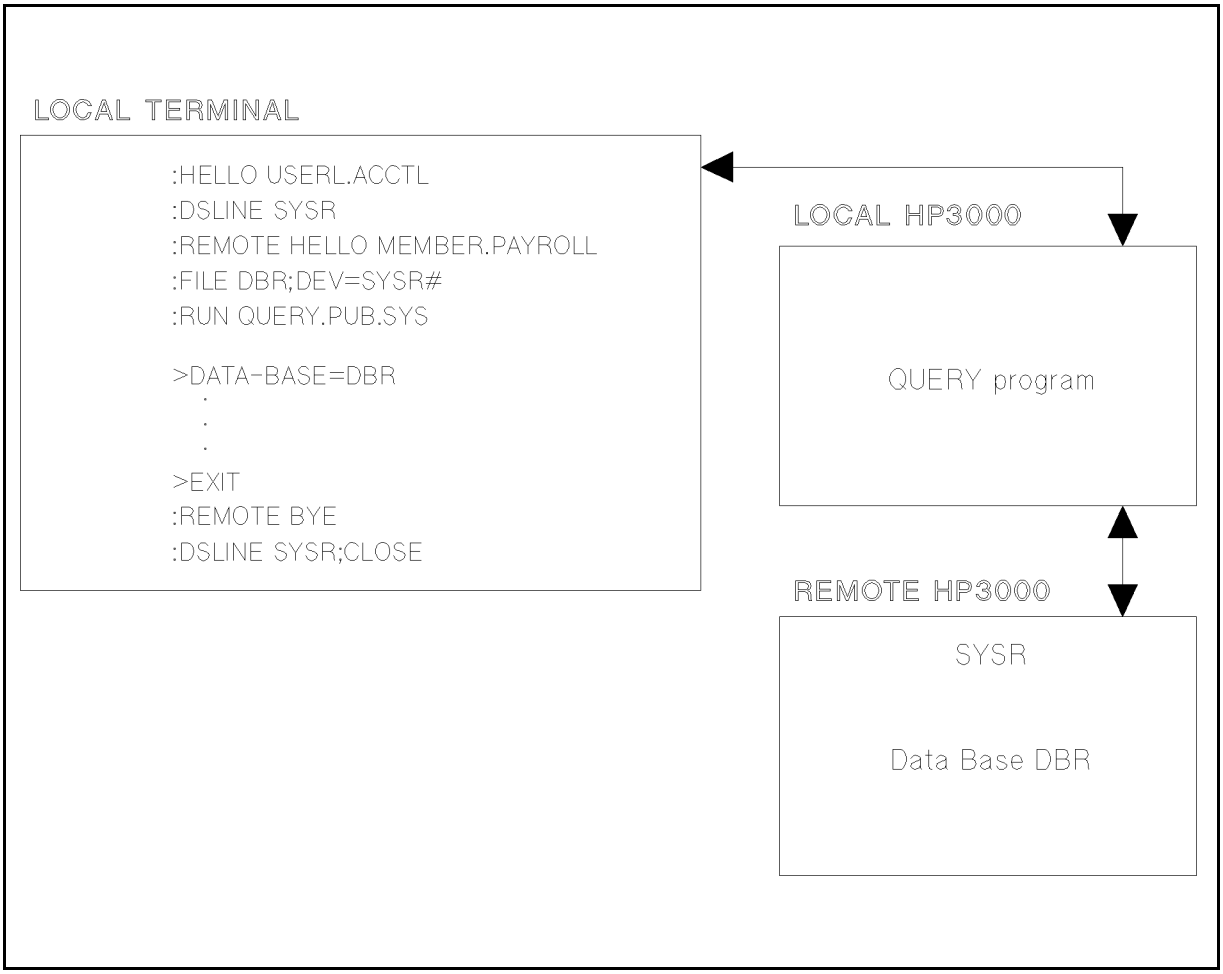


Figure C-2. Using Method 2

When using NS, you must have previously issued a :FILE command specifying the remote location of the file, to access a remote data base and run QUERY. In the example above, the user would enter:

```
:HELLO USERL.ACCTL
:DSLINE SYSR
:FILE DBR=DBX:SYSR
:REMOTE HELLO MEMBER.PAYROLL
:RUN QUERY.PUB.SYS
```

METHOD 3: USING A DATA-BASE-ACCESS FILE

The third method uses a data-base-access file to define the remote data base. When using the DEFINE or DATA-BASE= command, you enter the name of a data-base-access file rather than a data base name. Refer to the *IMAGE Reference Manual* for information on creating a data-base-access file. In this method, QUERY automatically issues a REMOTE BYE and closes the DSLINE when you enter the QUERY EXIT command.

Figure C-3 illustrates Method 3 using DS. The user logs on to the local system, runs QUERY and specifies a data-base-access file name with the DATA-BASE= command. The data-base-access file contains the name of a remote data base and the commands to establish a communications link and remote session to access the data base DBR that resides on the remote system. Record 1 of the data-base-access file specifies the remote system on which the data base resides and the device class of the disc (DISCA) on which it resides. Record 2 specifies the Distributed Systems (DS line) and Record 3 indicates the remote username and account name which is to be used to establish a remote session for USERL.ACCTL.

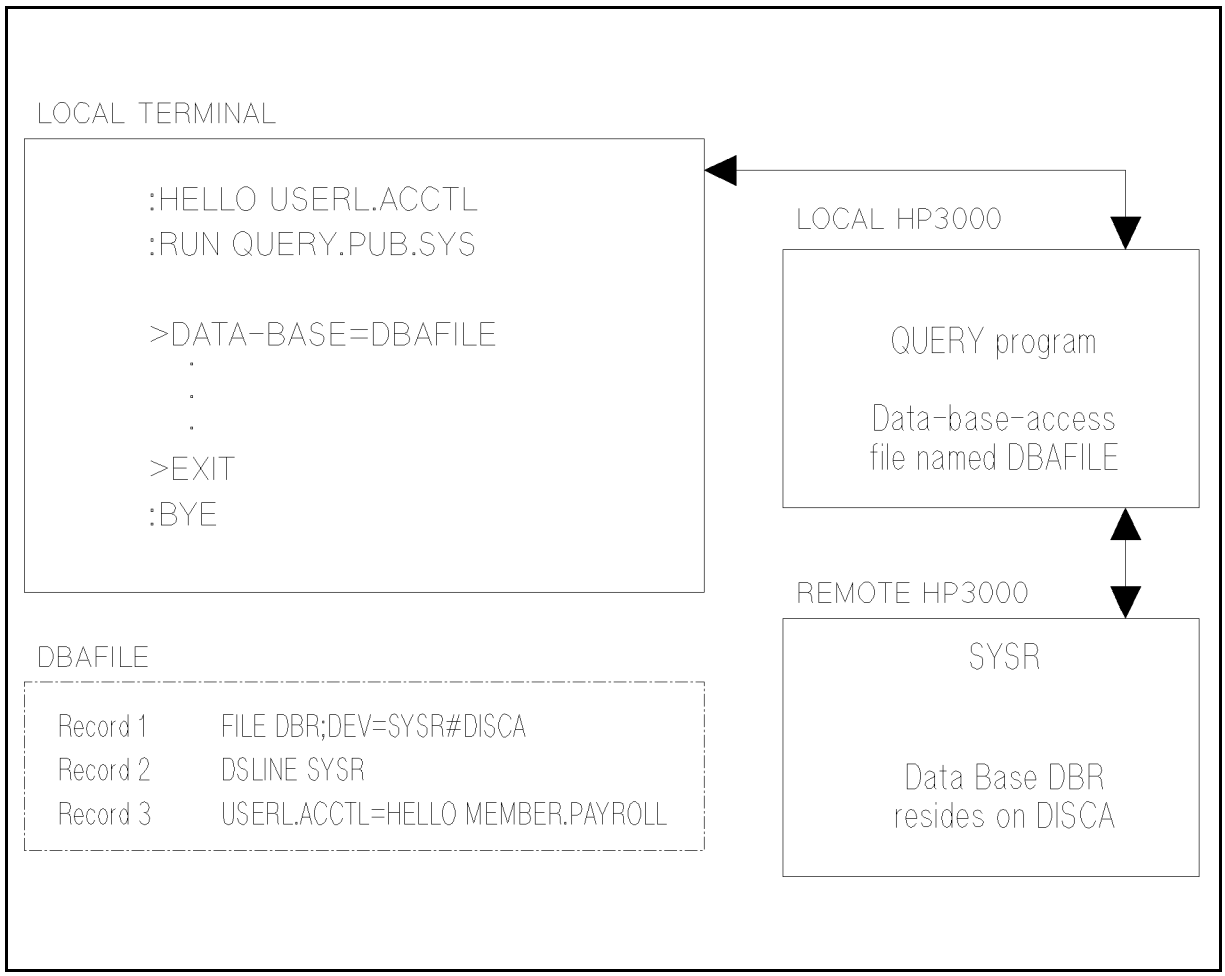


Figure C-3. Using Method 3

Note IMAGE verifies the content of the data-base-access file and will accept only the DS syntax for the :DSLINE and :FILE commands. This is due, in part, to the IMAGE restriction on the size of the dsdevice name to 8 characters. If you are using NS, input the device class name or logical device number in the first two records of the data-base-access file.

D

NATIVE LANGUAGE SUPPORT

Native Language Support (NLS) features can be accessed in QUERY to retrieve data which meet user-defined selection criteria, and to sort data according to native language collating sequences. The user must know what the user language in QUERY is, how the language is specified, how the language affects the output, and how to determine which language is being used.

IMAGE data bases have a language attribute that describes the collating sequence used in sorted chains and locking. This language attribute does not affect QUERY operation.

Although QUERY commands are in English, the output will be sorted and formatted according to the QUERY user's language. The language of the data base may determine the data sequence while using QUERY passively for retrieval (e.g., FIND). When data is being sorted or formatted by QUERY, the user's language will determine the ordering and formatting of the data.

For example, in a French data base with a QUERY user's language of Danish, data items in a sorted chain might be retrieved according to the French collating sequence; but the sorting or formatting is done according to Danish criteria.

The user can specify the QUERY user's language by:

- Using an MPE command:

  `:SETJCW NLUSERLANG =` *langnum* (Default is NATIVE-3000)

- Using a QUERY command:

  `>LANGUAGE=` *langnum* (Default is NLUSERLANG)

  or

  `>LANGUAGE=` *langname* (Default is NLUSERLANG)

For example, if the user's language is French, the QUERY command is:

```
>LANGUAGE= 7
```

or

```
>LANGUAGE= FRENCH
```

Or the MPE Job Control Word NLUSERLANG may be used:

```
:SETJCW NLUSERLANG = 7
```

The LANGUAGE= command always overrides NLUSERLANG. If neither option is used to specify the user's langauage, QUERY assumes LANGUAGE=0 (Native-3000). NATIVE-3000 is the default, which ensures backward compatibility. When the user's language is NATIVE-3000, QUERY performs as it did before NLS features were available.

QUERY allows access to more than one data base at the same time. This means that more than one data base language attribute may be active at the same time. In any case,

upshifting, collating, range selection, formatting, or sorting is dependent on the QUERY user's language specified by the user via the JCW NLUSERLANG or the LANGUAGE= command.

EFFECTS OF NLS ON QUERY

NLS can affect QUERY in upshifting data, range selection, data format, real number conversions, and sorted lists and numeric data editing in REPORT.

Upshifting Data (Type U Items)

QUERY upshifts commands and the data of type U items. QUERY commands are upshifted according to NATIVE-3000. Data is upshifted according to the user's language for UPDATE ADD, UPDATE REPLACE, FIND, LIST, MULTIFIND, and SUBSET.

Range Selection

QUERY compares character data according to the user's language when using FIND, LIST, MULTIFIND, or SUBSET. The MATCHING feature (in FIND and MULTIFIND commands) is only valid when LANGUAGE=0 (NATIVE-3000). QUERY will display an error message if MATCHING is used in an interactive mode, and will abort the session in a batch mode.

Date Format

DATE is a reserved word in the REPORT command which provides the system date. It is formatted according to the user's language.

Real Number Conversions

In the commands REPORT and LIST the output is formatted according to the user's language. For example, 123.45 in NATIVE-3000 becomes 123,45 in FRENCH.

Sorted Lists in REPORT

QUERY sorts type U or X items in a REPORT according to the collating sequence of the user's language.

Numeric Data Editing in REPORT

QUERY converts the data edited using the NATIVE-3000 edit mask (using the period as a decimal point and a comma as a thousands separator) to the corresponding characters in the user's language.

The commands in Table D-1 are used to obtain language-dependent information.

Table D-1. Commands for Language-Dependent Information

COMMAND	LANGUAGE-DEPENDENT INFORMATION
`HELP LANGUAGE=`	Explains the LANGUAGE= command function, format and parameters.
`LANGUAGE=`	Specifies the user language.
`SHOW LANGUAGE`	Displays the QUERY user's language.
`FORM`	Displays the data base language attribute.

E

SELF-DESCRIBING FILES

A self-describing file is created when the SAVE command is executed. A file is considered to be self-describing when the user labels contain information about the format of the file. A self-describing (SD) file is composed of two or more user labels and data.

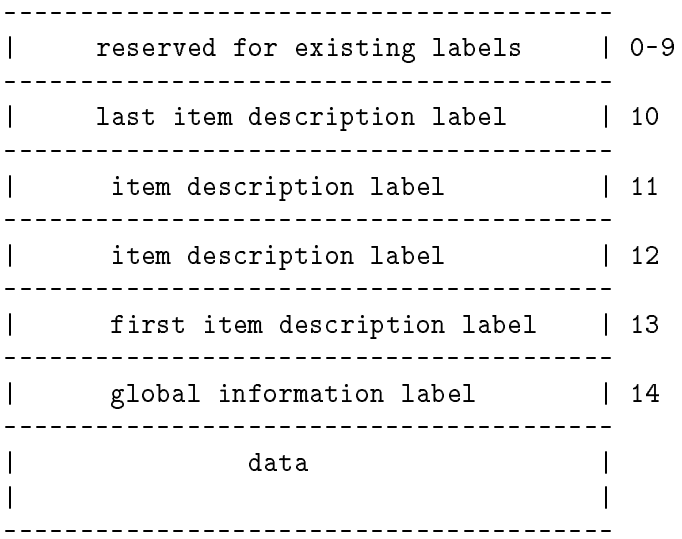
FILE DATA

Each record in the file holds an IMAGE entry that qualified from a FIND or MULTIFIND retrieval. The number of records in the file is the same as the number of entries retrieved and the size of each record is the same as the size of the retrieved entry. In a FIND, the records are the size of the entries in a single data set. In a MULTIFIND, the records are the size of the entries in a compound data set. A compound data set entry is the combined size of the entries in the single data sets of which it is comprised (specified in the JOIN command).

The data in the file is stored in the same form as the data in the IMAGE data base. Numeric data is stored in binary form and must be converted to ASCII before it can be displayed.

USER LABELS

There are two types of user labels which both contain descriptive information. One type contains global information about the file and the data it contains. The other type contains specific information about each item in the data set entry or compound data set entry in the file. Each user label is 128 words in length and is placed in the file as follows:



Labels are in "reverse" order in the file. That is the last label is the global information label, the second to last label is the first item description label. An item description label contains one or more item descriptions and there must be at least one item description label.

When the file is built, space is reserved for pre-existing user labels. Pre-existing labels may have been placed in the file by an application program. If another application program converts an existing file with user labels into an SD file, the existing labels are not overwritten unless there are more than ten. Pre-existing user labels begin at label 0. As a result, the last item description label always occupies label 10 in the SD file.

Global Information Label Format

The information in the global information label resides in the following order:

version	*length*	*#items*	*#labels*	*#items/ labels*	size

version is the the version of the SD Intrinsics that created the file. The standard HP version format is followed (AA.NN.NN). (8 characters, right justified)

length is the length of the records in bytes. This corresponds to the size of the IMAGE entry. (integer)

#items is the number of items in each file record. This corresponds to the number of items in the IMAGE entry. (integer)

#labels is the number of item description user labels. (integer)

#items/labels is the number of item descriptions that fit into one item description label. Each label can hold INTEGER (128/size). (integer)

size is the size of one item description in words. (integer)

Item Description Label Format

The information in the item description label resides in the following order:

item name	*item type*	*item offset*	*item length*	*reserved space*

item name is the name of the item. (16 characters, left justified)

item type is one of the following codes indicating the type of item. (integer)

1 = ASCII (type U and X)
2 = ASCII numeric in free form
3 = signed integer (type I)
4 = floating point real (type R)
5 = packed decimal (type P)
6 = COBOL computational (type J)
7 = unsigned integers (type K)
8 = zoned decimal (type Z)
9 = unused

10 = IMAGE compound item

The item types in parentheses correspond to IMAGE data types.

Note: Type 2 refers to ASCII representations of numbers in free format. This includes 20, 45.7, 1.002E-10, -201.45.

item offset is the offset of the item in the file record in bytes. This offset is relative to the beginning of the beginning of a record where the first byte has an offset of zero, the second has an offset of one, and so on. (integer)

item length is the length of the item in bytes. This distinguishes the long and short forms of integers and reals. (integer)

reserved space is reserved for future enhancements to SD files. Four words are reserved.

ITEM NAMES

All item names in a self-describing file must be unique unless all the duplicate items are in data item equivalences. (Refer to the JOIN command for more information on data item equivalences.)

Entries retrieved with a FIND have unique item names because they are retrieved from a single data set. However, entries retrieved with a MULTIFIND may have duplicate item names because they are retrieved from more than one data set. The duplicate items must have been used in the JOIN data item equivalence(s). An exception is made because the duplicate items are of the same value although they may be of different data types.

The first occurence of the item in the compound data set entry is the one recorded in the item description. Other occurences are skipped because the item has already been described. The data in the file contains all occurences of the data item even though the item description appears only once. The offset reflects the first location in the record where the data item value can be accessed.

Refer to example 2 for a self-describing file created from a MULTIFIND where the JOIN item names are identical.

Examples

Example 1 - Single Data Set Retrieval

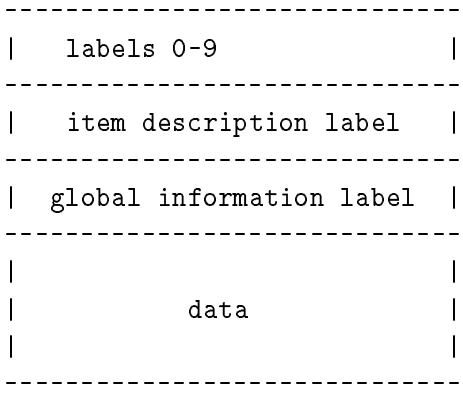
Given the following IMAGE data base:

```
BEGIN DATA BASE SD;
PASSWORDS:
ITEMS:
  NAME,  U10;
  OCCUP, U16;
  CITY,  U10;
  STATE, I2;   << contains a numeric code for each state >>
SETS:
  NAME: SET1, D;
  ENTRY: NAME, OCCUP, CITY, STATE;
  CAPACITY: 11;
END.
```

and the following FIND and SAVE commands:

```
>FIND STATE=2 OR STATE=3
USING SERIAL READ
4  ENTRIES QUALIFIED
>SAVE SAVEFILE
```

the SD file, SAVEFILE, appears as follows:



where the global information label contains:

(version)	(length)	(#items)	(#labels)	(#items/label)	(size)
A.00.00	38	4	2	8	15

and the item description label contains:

(name)	(type)	(offset)	(length)	(unused)
NAME	1	0	10	0
OCCUP	1	10	16	0
CITY	1	26	10	0
STATE	3	36	2	0

and the data in the file is (binary data is represented as periods):

```
M FULLER  ENGINEER        CUPERTINO ..
H THOREAU ENGINEER        SAN DIEGO ..
W EMERSON ENGINEER        FT COLLINS ..
G SAND    TECH WRITER     CUPERTINO ..
```

Example 2 - Compound Data Set Retrieval

Given the following IMAGE data base:

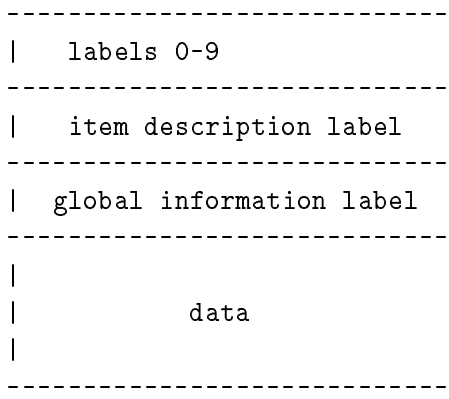
```
BEGIN DATA BASE SD2;
PASSWORDS:
ITEMS:
```

```
    NAME,   U10;
    OCCUP,  U16;
    CITY,   U10;
    STATE,  I1;
    NAME2,  U10;
    OCCUP2, U16;
    CITY2,  U10;
  SETS:
    NAME: SET1, D;
    ENTRY: NAME, OCCUP, CITY, STATE;
    CAPACITY: 11;

    NAME: SET2, D;
    ENTRY: NAME2, OCCUP2, CITY2, STATE;
    CAPACITY: 11;
  END.
```

and the following JOIN, MULTIFIND, and SAVE commands:

```
>JOIN SET1.STATE TO SET2.STATE
>MU STATE=2 OR STATE=3
USING SORT/MERGE
USING SERIAL READ
10  COMPOUND ENTRIES QULIFIED
>SAVE SAV2FILE
```

the SD file, SAV2FIL would look like this:



where the global information label contains:

(version)	(length)	(#items)	(#labels)	(#items/label)	(size)
A.00.00	74	7	2	8	15

and the item description label contains:

(name)	(type)	(offset)	(length)	(unused)
NAME	1	0	10	0

OCCUP	1	10	16	0
CITY	1	26	10	0
STATE	3	36	2	0
NAME2	1	38	10	0
OCCUP2	1	48	16	
CITY2	1	64	10	0

and the data in the file is (binary data is represented as periods):

```
M FULLER  ENGINEER        CUPERTINO ..M FULLER2 ENGINEER        CUPERTINO
M FULLER  ENGINEER        CUPERTINO ..H THOREAU2ENGINEER        CUPERTINO
M FULLER  ENGINEER        CUPERTINO ..G SAND2   TECH WRITER     CUPERTINO
H THOREAU ENGINEER        SAN DIEGO ..M FULLER2 ENGINEER        CUPERTINO
H THOREAU ENGINEER        SAN DIEGO ..H THOREAU ENGINEER        CUPERTINO
H THOREAU ENGINEER        SAN DIEGO ..G SAND2   TECH WRITER     CUPERTINO
G SAND    TECH WRITER     CUPERTINO ..M FULLER2 ENGINEER        CUPERTINO
G SAND    TECH WRTIER     CUPERTINO ..H THOREAU2ENGINEER        CUPERTINO
G SAND    TECH WRITER     CUPERTINO ..G SAND2   TECH WRITER     CUPERTINO
W EMERSON ENGINEER        FT COLLINS..W EMERSON2ENGINEER        FT COLLINS
```

F

USER-DEFINED PROCEDURES

Procedures

This appendix documents a feature of the QUERY subsystem which is used primarily by the programmer or data base administrator. QUERY allows you to specify your own procedure(s) which will enable a report to read or write from another data base, data set, or file, modify registers, or perform other specialized tasks not provided by QUERY. The name of the user-defined procedure cannot be the name of any data item in the data sets referenced by the retrieval command.

User-defined procedures can be called from either a header, group, total, or detail statement within the REPORT command, and can be written in SPL, COBOL, FORTRAN or PASCAL. Note that programming errors in user-defined procedures will cause QUERY to terminate. The procedure must be stored in one of the segmented libraries. These libraries are searched in the following order:

```
group
account
system
```

To call your defined procedure, replace the print element with the name of the procedure, and replace the print position with a slash (/). Seventeen parameters are automatically passed to the procedure by QUERY. You may provide a single integer parameter for each procedure. To specify a value for a particular call of the procedure, the value should be placed in parenthesis after the name of the procedure.

report statement type,procedure name [(*user parameter*)],/,[*statement& parameter*]

The following statement parameters are allowed with a REPORT statement containing a call to a user-defined procedure:

```
SPACE A
SPACE B
SKIP A
SKIP B
```

For example:

```
REPORT
H1,PROC1(5),/,SPACE A3
H1,ITEM3,50
D1,PROC2,/
D1,ITEM1,10
D2,ITEM2,10
END
```

The following is not allowed:

```
EDIT masks
ADD
AVERAGE
COUNT
```

If you specify an option that is not allowed, QUERY will issue the following message.

```
A SPECIFIED OPTION IS NOT ALLOWED FOR USER PROCEDURES
```

There are eight parameters that you can set. These are: CALLAGAIN, PAGE'EJECT, LINEBUF, REGARR, USERSTOR, USERPARAM, DATA'BASE'NAME, and DATA'SET'NAME. The other nine parameters are values returned by QUERY. Within your procedure, you must declare parameters with the same or different names in the following order:

```
LOGICAL           CALLAGAIN,            1 word
                  PAGE'EJECT;           1 word
INTEGER ARRAY     BASE'IDS,           130 words
                  DBBUF,             2051 words
                  DSET'LIST            53 words
                  DATA'BASE'LIST       53 words
                  LINEBUF,             69 words
                  REGARR,             150 words
                  USERSTOR,            64 words
                  USERPARAM,            1 word
                  DATA'BASE'NAME,      13 words
                  DATA'SET'NAME,        8 words
INTEGER           CALL'NUM,             1 word
                  LINES'LEFT,           1 word
                  PAGENO,               1 word
                  NUM'DSETS             1 word
                  NUM'DBASES;           1 word
```

* The following parameters can be set by you.

Parameters

**CALLAGAIN* is a flag telling QUERY to call this procedure again. QUERY will stop calling the procedure only when this parameter is returned FALSE. This flag is for the particular report statement being processed. For example, if `D1,PROC,/ + CALLAGAIN = T`, then the procedure will be called again from this detail statement until *CALLAGAIN* is set to false. The default value is FALSE.

**PAGE'EJECT* is a flag telling QUERY to perform a page eject upon return. If you determine by the *LINES'LEFT* parameter that there are not enough lines left on the page to perform the desired task, you can assign TRUE to this parameter. In this case, the *CALLAGAIN* parameter may also be returned and the procedure will be executed on the new page following headers (if any). This parameter can also be used if a page eject is desired after execution. The default value is FALSE.

BASE'IDS is the data base name of the data base(s) currently being accessed. Each base id is 26 bytes (13 words) long for a total of 10 base id's.

Each name is preceded by the *BASE'ID* number (1 word) assigned by IMAGE.

DBBUF is an array which holds the values of the data items to which you have access for a specific data set for the current entry or compound entry being output. The values of the data set are placed in *DBBUF* as follows:

If the FIND command, or a SUBSET of a FIND command, was used to select the entries for reporting, then the data set that the FIND or SUBSET command referenced will be used.

If the MULTIFIND command or a SUBSET of a MULTIFIND was used, then the first data set mentioned in the JOIN command is used.

On QUERY version B.01.10 and later, the length of the entry being passed to you is placed in the last word of this array (word 2051). If the entry is a null entry, from a MULTIFIND following a join containing an @ sign, this length will be set to zero and *DBBUF* will be filled with ASCII nulls.

If you want a different entry from any data set mentioned in the FIND, SUBSET, or JOIN command, set the *DATA'BASE'NAME* and *DATA'SET'NAME* parameters with the appropriate names and set the *CALLAGAIN* parameter to TRUE. The items from the desired data set will be returned to your procedure from QUERY.

DSET'LIST contains the data set numbers of those data sets accessed by the FIND, MULTIFIND, or SUBSET command.

DATA'BASE'LIST contains the position of the data base name in the parameter *BASE'IDS* for the corresponding entry in the parameter *DSET'LIST*. For example:

A zero (0) in *DATA'BASE'LIST*(2) means that the data set in *DSET'LIST*(2) belongs to the first data base named in *BASE'IDS*.

**LINEBUF* is the buffer that QUERY uses to build each line of REPORT. For each line of output that is generated, those statements corresponding to that line which contain a print element or which call a user procedure that modifies *LINEBUF*, operate cummulatively on *LINEBUF* to create the line of output. If *LINEBUF* has been changed by the procedure, it will print when all report statements of that level have been processed. (For example, it will be printed when all D1 statments have been processed.) On QUERY versions prior to C.00.00, LINEBUFF will print only if there is another statement of the same level that contains a print element.

**REGARR* contains the 30 registers, with 5 words allocated for each register. You must know the types of the registers that you access. The types used by QUERY are:

`P20:`	uses all five words (right-justified)
`R2:`	uses the leftmost 2 words (left-justified)
`R4:`	uses the leftmost 4 words (left-justified)

The section on registers in the REPORT command description explains how QUERY determines the type of a register.

**USERSTOR* is a global scratch area for user data which is shared by all of your procedures that are referenced in any one report. Not initialized.

**USERPARAM* is where the value of your parameter is stored. One use of this parameter might be to indicate where in the output buffer to place the value. (For example, `D1,PROC,/;D1,PROC(20),/; ...`)

**DATA'BASE'NAME* is set with the correct data base name if data item values are needed from a data set(s) other than the default data set(s), (see above parameter *DBBUF*) or data item values are needed for data items not mentioned in the report. The name should be upper case, left-justified and, if necessary, qualified with group and account. The rest of the array should be filled with blanks. The parameter *DATA'SET'NAME* must also be set. If the specified name is invalid, QUERY will give the message:

```
NO RETRIEVAL WAS MADE FROM THE DATA BASE XX,
WHICH WAS NAMED IN A USER PROCEDURE.
```

**DATA'SET'NAME* is set with the correct data set name if data item values are needed from a data set(s) other than the default data set(s) (see above parameter *DBBUF*), or data item values are needed of data items other than those mentioned in the report. The name should be upper case and left-justified and the rest of the array should be filled with blanks. The parameter *DATA'BASE'NAME* must also be set. If the specified name is invalid, QUERY will give the following error message:

```
THE DATA SET XX, NAMED IN A USER PROCEDURE,
IS NOT IN DATA BASE YY
```

CALL'NUM is the number of times that a procedure has been called from the same level of report statments. The first time the procedure is called the number will be 1. `CALL'NUM` is reset to 1 after all the report statements of a particular level are processed. For example (assume *CALLAGAIN* is always FALSE):

```
D1,PROC1,1   << call'num = 1 >>
D1,PROC2,1   << call'num = 2 >>
D1,ITEM,10
D2,ITEM,10
D2,PROC3,1   << call'num = 1 >>
```

LINES-LEFT is the number of lines that are left on the page.

PAGENO tells which page is currently being output.

NUM'DSETS is the number of data sets that are accessed by REPORT.

NUM'DBASES is the number of data bases open by the current user of QUERY.

Examples

The following examples show two user-defined procedures: MPROC and QPROC. MPROC is shown in four languages and QPROC is shown in two languages.

MPROC Procedure

The following examples show MPROC in SPL, COBOL, PASCAL and FORTRAN. The data base, retrieval, and report are shown for understanding the context of the use of the procedure.

The data base is defined as follows:

```
BEGIN DATA BASE ACCTS;
PASSWORDS:
ITEMS:
   ACCT-NUM,   X6;
   ORDER-DATE, I2;    << DATES ARE STORED AS DOUBLE INTEGERS, >>
   SHIP-DATE,  I2;    << IE 84140 IS 1984, 140TH DAY IN YEAR  >>
   DEPT,       U4;
   CARRIER,    U4;
SETS:
   NAME: ACCT-MAST, A;
   ENTRY: ACCT-NUM (2);
   CAPACITY: 11;

   NAME: ORDER-D, D;
   ENTRY: ACCT-NUM (!ACCT-MAST),
          ORDER-DATE,
          DEPT;
   CAPACITY: 11;

   NAME: SHIP-D, D;
   ENTRY: ACCT-NUM (!ACCT-MAST),
          CARRIER,
          SHIP-DATE;
   CAPACITY: 11;
END.
```

The retrieval and report are defined as follows:

```
>JOIN ORDER-D.ACCT-NUM TO SHIP-D.ACCT-NUM
>MU ALL
USING SERIAL READ
14 COMPOUND ENTRIES RETRIEVED
>XEQ MULTIREP
```

The XEQ file, MULTIREP, contains:

```
REPORT
H1,"REPORT ON ALL ACCOUNT DATES",45,SPACE A2
H2,"ACCOUNT",8
H2,"DEPT",14
H2,"DATE OF ORDER",35
H2,"SHIPPING DATE",55
H2,"CARRIER",65
H3,"-------",8
H3,"----",14
H3,"-------------",35
H3,"-------------",55,SPACE A2
H3,"-------",65
S1,ORDER-DATE
S2,ACCT-NUM
```

```
D1,ACCT-NUM,8,E1
D1,":",9
D1,DEPT,14
D1,CARRIER,65
D1,MPROC,/
G2,"    ",5
E1,"XXX-XXX"
END
```

The output is:

```
                REPORT ON ALL ACCOUNT DATES

ACCOUNT  DEPT        DATE OF ORDER       SHIPPING DATE   CARRIER
-------  ----        -------------       -------------   -------

010-666: 008     WED, DEC  5, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC  5, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC  5, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 12, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 12, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 12, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 26, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 26, 1984   THU, DEC 27, 1984      UPS
010-666: 008     WED, DEC 26, 1984   THU, DEC 27, 1984      UPS

041-321: 003     WED, DEC 12, 1984   THU, DEC 13, 1984      UPS

055-433: 005     FRI, DEC 14, 1984   FRI, DEC 28, 1984      UPS
055-433: 005     FRI, DEC 14, 1984   FRI, DEC 28, 1984      UPS
055-433: 005     WED, DEC 26, 1984   FRI, DEC 28, 1984      UPS
055-433: 005     WED, DEC 26, 1984   FRI, DEC 28, 1984      UPS
```

SPL - MPROC Procedure

```
<<-------------------------------------------------------->>
<< HAVING USLINIT AND SUBPROGRAM WILL GENERATE A WARNING,>>
<< YOU WILL NOT WANT TO INITIALIZE THE USL FILE IF YOU   >>
<< ARE COMPILING DIFFERENT SUBPROGRAMS INTO IT.          >>
<<-------------------------------------------------------->>
$CONTROL MAP, USLINIT, SUBPROGRAM
$CONTROL SEGMENT=MULTISEG
<<-------------------------------------------------------->>
<<                                                       >>
<<                      MPROC                            >>
<<                                                       >>
<< EXAMPLE OF A SPL USER-DEFINED PROCEDURE, USED WITH A >>
<< REPORT STATEMENT.                                     >>
<<                                                       >>
<<-------------------------------------------------------->>
<< THIS PROCEDURE TAKES A JULIAN DATE AND CONVERTS IT   >>
<< TO "DAY, MONTH, YEAR" FORMAT BY CALLING THE MPE      >>
<< INTRINSIC FMTCALENDAR.                                >>
<<-------------------------------------------------------->>
BEGIN

<<----------------------------------------->>
<<          MPROC DECLARATION              >>
<<----------------------------------------->>

PROCEDURE MPROC (CALLAGAIN, PAGE'EJECT, BASE'IDS, DBBUF,
                 DSET'LIST, DATA'BASE'LIST, LINEBUF, REGARR,
                 USERSTOR, USERPARAM, DATA'BASE'NAME,
                 DATA'SET'NAME, CALL'NUM, LINES'LEFT, PAGENO,
                 NUM'DSETS, NUM'DBASES);
```

```
LOGICAL CALLAGAIN, PAGE'EJECT;
INTEGER ARRAY BASE'IDS, DBBUF, DSET'LIST, DATA'BASE'LIST,
              LINEBUF, REGARR, USERSTOR, USERPARAM,
              DATA'BASE'NAME, DATA'SET'NAME;
INTEGER       CALL'NUM, LINES'LEFT, PAGENO, NUM'DSETS, NUM'DBASES;

BEGIN

BYTE ARRAY FMTDATE (0:16),
           BLINEBUF (*) = LINEBUF;

DOUBLE POINTER D'DATE;

DOUBLE ARRAY D'DAY (0:0),
             D'YEAR (0:0);

INTEGER ARRAY DAY (*) = D'DAY,
              YEAR (*) = D'YEAR;

DOUBLE HOLD;

INTEGER JDATE;

INTRINSIC FMTCALENDAR;

<<-------------------------------------------------------->>
<< THERE ARE TWO DATES IN THE DATA BASE TO BE FORMATTED.  >>
<< ONE IS IN THE ORDER-D DATA SET, THE OTHER IS IN THE    >>
<< SHIP-D DATA SET.  BECAUSE THE SETS HAVE BEEN JOINED,   >>
<< IT IS POSSIBLE TO GET BOTH DATES AND FORMAT THEM.      >>
<< NOTE: IT IS THE APPLICATION'S RESPONSIBILITY TO KNOW    >>
<< WHERE IN THE DATA BUFFER THE VALUE IS LOCATED.         >>
<<-------------------------------------------------------->>
<<-------------------------------------------------------->>
<< THE DATE IS IN A TWO WORD INTEGER AND NEEDS TO BE RE-  >>
<< FORMATTED BEFORE PASSING TO FMTCALENDAR.               >>
<<-------------------------------------------------------->>
<<-------------------------------------------------------->>
<< IF CALL'NUM IS ONE THEN WE CAN ASSUME THE ENTRY IS     >>
<< FROM THE ORDER-D DATA SET, SINCE IT IS THE FIRST DATA  >>
<< SET MENTIONED IN THE JOIN COMMAND.  IF CALL'NUM IS NOT >>
<< ONE THEN WE CAN ASSUME THAT THE PROCEDURE HAS BEEN     >>
<< CALLED AGAIN, AND WE NOW HAVE THE ENTRY FROM THE       >>
<< SHIP-D DATA SET.                                       >>
<<-------------------------------------------------------->>

IF CALL'NUM =1 THEN                          << FIRST DATE >>
   @D'DATE := @DBBUF(3)
ELSE                                         << SECOND DATE >>
   @D'DATE := @DBBUF(5);

D'YEAR := D'DATE / 1000D;                   << ISOLATE YEAR >>
HOLD := D'YEAR * 1000D;
D'DAY := D'DATE - HOLD;
JDATE := DAY (1);                            << DAY IN BITS 0-6 >>
JDATE.(0:7) := YEAR (1);                    << YEAR IN BITS 7-15 >>
FMTCALENDAR (JDATE, FMTDATE);

<<-------------------------------------------------------->>
<< PUT THE FORMATTED DATE INTO THE OUTPUT BUFFER USED BY  >>
<< QUERY FOR THE REPORT OUTPUT LINE.  AGAIN, CALL'NUM IS  >>
<< USED TO DETERMINE WHERE IN THE OUTPUT BUFFER TO PLACE  >>
<< THE VALUE.  ALSO, IF CALL'NUM IS ONE THEN WE NEED TO   >>
<< SET THE DATA BASE NAME, THE DATA SET NAME, AND THE     >>
<< CALLAGAIN PARAMETERS IN ORDER TO GET THE ENTRY FROM    >>
```

```
<< THE SHIP-D DATA SET TO FORMAT THE SHIPPING DATE.       >>
<<--------------------------------------------------------->>

IF CALL'NUM = 1 THEN              << ORDER-D DATA SET >>
   BEGIN
   MOVE BLINEBUF(18) := FMTDATE, (17);
   MOVE DATA'BASE'NAME := "ACCTS        ";
   MOVE DATA'SET'NAME := "SHIP-D  ";
   CALLAGAIN := TRUE;
   END

ELSE                               << SHIP-D DATA SET >>
   BEGIN
   MOVE BLINEBUF(38) := FMTDATE, (17);
   CALLAGAIN := FALSE;
   END;

END;

END.
```

COBOL - MPROC Procedure

```
$CONTROL USLINIT, SUBPROGRAM, MAP, DYNAMIC

 IDENTIFICATION DIVISION.
  PROGRAM-ID. MPROC.
  AUTHOR. HP.

*--------------------------------------------------------*
*                                                        *
*                      MPROC                             *
*                                                        *
* EXAMPLE OF A COBOL USER-DEFINED PROCEDURE, USED WITH *
* A REPORT STATEMENT.                                    *
*                                                        *
* THIS MUST BE COMPILED WITH THE COBOLII COMPILER.    *
*                                                       *
*--------------------------------------------------------*
* THIS PROCEDURE TAKES A JULIAN DATE AND CONVERTS IT   *
* TO "DAY, MONTH, YEAR" FORMAT BY CALLING THE MPE      *
* INTRINSIC FMTCALENDAR.                                 *
*--------------------------------------------------------*

 ENVIRONMENT DIVISION.


 DATA DIVISION.

  WORKING-STORAGE SECTION.
  01  FMTDATE-REC.
      05 FMTDATE OCCURS 17 PIC X.
  01  I-DAY               PIC S9(4) COMP.
  01  YEAR                PIC S9(4) COMP.
  01  HOLD                PIC S9(9) COMP.
  01  D-JDATE             PIC S9(9) COMP.
  01  JDATE REDEFINES D-JDATE.
      05  I-JDATE OCCURS 2 PIC S9(4) COMP.
  01  INDX                PIC S9(4) COMP.


  LINKAGE SECTION.
  01 CALLAGAIN            PIC S9(4)   COMP.
  01 PAGE-EJECT           PIC S9(4)   COMP.
  01 BASE-IDS.
```

```
     05 BASE-ID1          PIC S9(4)   COMP.
     05 BASE-NAME1        PIC X(24).
     05 BASE-ID2          PIC S9(4)   COMP.
     05 BASE-NAME2        PIC X(24).
     05 BASE-ID-OTHERS    PIC X(208).
  01 DBBUF                PIC X(4102).
  01 DBBUF-2 REDEFINES DBBUF.
     05 FILLER            PIC X(6).
     05 DBBUFD OCCURS 2 PIC S9(9) COMP.
     05 FILLER            PIC X(4088).
  01 DSET-LIST.
      05 DSET       OCCURS 100 TIMES PIC S9(4) COMP.
  01 DATA-BASE-LIST.
      05 DBASE      OCCURS 100 TIMES PIC S9(4) COMP.
  01 LINEBUF.
     05 LINEBUF-70        PIC X(70).
     05 LINEBUF-OTHER     PIC X(66).
  01 BLINEBUF1 REDEFINES LINEBUF.
     05 FILLER             PIC X(18).
     05 BA-LINEBUF1        PIC X(17).
     05 FILLER             PIC X(101).
  01 BLINEBUF2 REDEFINES LINEBUF.
     05 FILLER             PIC X(38).
     05 BA-LINEBUF2        PIC X(17).
     05 FILLER             PIC X(81).
  01 REGARR.
      05  REG   OCCURS 30 TIMES  PIC S9(18) COMP-3.
  01 USERSTOR.
     05 USERSTOR-70       PIC X(70).
     05 USERSTOR-OTHER    PIC X(58).
  01 USERPARAM            PIC S9(4)   COMP.
  01 DATA-BASE-NAME       PIC X(26).
  01 DATA-SET-NAME        PIC X(16).
  01 CALL-NUM             PIC S9(4)   COMP.
  01 LINES-LEFT           PIC S9(4)   COMP.
  01 PAGENO               PIC S9(4)   COMP.
  01 NUM-DSETS            PIC S9(4)   COMP.
  01 NUM-DBASES           PIC S9(4)   COMP.

*---------------------------------------------------------*
*                 MPROC PROCEDURE DIVISION                *
*---------------------------------------------------------*

 PROCEDURE DIVISION USING CALLAGAIN, PAGE-EJECT, BASE-IDS,
     DBBUF, DSET-LIST, DATA-BASE-LIST, LINEBUF, REGARR,
     USERSTOR, USERPARAM, DATA-BASE-NAME, DATA-SET-NAME,
     CALL-NUM, LINES-LEFT, PAGENO, NUM-DSETS, NUM-DBASES.

 BEGIN-MPROC.

*---------------------------------------------------------*
* THERE ARE TWO DATES IN THE DATA BASE TO BE FORMATTED.  *
* ONE IS IN THE ORDER-D DATA SET, THE OTHER IS IN THE    *
* SHIP-D DATA SET.  BECAUSE THE SETS HAVE BEEN JOINED,   *
* IT IS POSSIBLE TO GET BOTH DATES AND FORMAT THEM.      *
* NOTE: IT IS THE APPLICATION'S RESPONSIBILITY TO KNOW    *
* WHERE IN THE DATA BUFFER THE VALUE IS LOCATED.         *
*---------------------------------------------------------*
*---------------------------------------------------------*
* THE DATE IS IN A TWO WORD INTEGER AND NEEDS TO BE RE-  *
* FORMATTED BEFORE PASSING TO FMTCALENDAR.               *
*---------------------------------------------------------*
*---------------------------------------------------------*
* IF CALL'NUM IS ONE THEN WE CAN ASSUME THE ENTRY IS     *
* FROM THE ORDER-D DATA SET, SINCE IT IS THE FIRST DATA  *
* SET MENTIONED IN THE JOIN COMMAND.  IF CALL'NUM IS NOT *
```

```
* ONE THEN WE CAN ASSUME THAT THE PROCEDURE HAS BEEN     *
* CALLED AGAIN, AND WE NOW HAVE THE ENTRY FROM THE       *
* SHIP-D DATA SET.                                       *
*--------------------------------------------------------*

 IF CALL-NUM = 1 THEN
    MOVE 1 TO INDX
 ELSE
    MOVE 2 TO INDX.

 DIVIDE DBBUFD(INDX) BY 1000 GIVING YEAR.
 MULTIPLY YEAR BY 1000 GIVING HOLD.
 SUBTRACT HOLD FROM DBBUFD(INDX) GIVING I-DAY.
 MULTIPLY YEAR BY 512 GIVING D-JDATE.
 ADD I-DAY TO D-JDATE.

 CALL INTRINSIC "FMTCALENDAR" USING I-JDATE(2), FMTDATE.

*--------------------------------------------------------*
* PUT THE FORMATTED DATE INTO THE OUTPUT BUFFER USED BY  *
* QUERY FOR THE REPORT OUTPUT LINE.  AGAIN, CALL'NUM IS  *
* USED TO DETERMINE WHERE IN THE OUTPUT BUFFER TO PLACE  *
* THE VALUE.  ALSO, IF CALL'NUM IS ONE THEN WE NEED TO   *
* SET THE DATA BASE NAME, THE DATA SET NAME, AND THE     *
* CALLAGAIN PARAMETERS IN ORDER TO GET THE ENTRY FROM    *
* THE SHIP-D DATA SET TO FORMAT THE SHIPPING DATE.       *
*--------------------------------------------------------*

 IF CALL-NUM = 1 THEN
    MOVE FMTDATE-REC TO BA-LINEBUF1
    MOVE "ACCTS                      " TO DATA-BASE-NAME
    MOVE "SHIP-D          " TO DATA-SET-NAME
    MOVE 1 TO CALLAGAIN
 ELSE
    MOVE FMTDATE-REC TO BA-LINEBUF2
    MOVE 0 TO CALLAGAIN.

 GOBACK.
```

COBOL Notes

The call to FMTCALENDAR will generate a 'Questionable' error because the second word of the I-JDATE array is being passed.

PASCAL - MPROC Procedure

```
(*--------------------------------------------------------*)
(* YOU WILL NOT WANT TO INITIALIZE THE USL FILE IF YOU   *)
(* ARE COMPILING DIFFERENT SUBPROGRAMS INTO IT.          *)
(*--------------------------------------------------------*)
$CODE_OFFSETS ON; TABLES ON; USLINIT; SUBPROGRAM$
$STANDARD_LEVEL 'HP3000'$
$SEGMENT 'MULTISEG'$
(*--------------------------------------------------------*)
(*                                                       *)
(*                      MPROC                            *)
(*                                                       *)
(* EXAMPLE OF A PASCAL USER-DEFINED PROCEDURE, USED      *)
(* WITH A REPORT STATEMENT.                              *)
(*                                                       *)
(* THE PROCEDURE CONTAINS COMMENT LINES WHICH ONLY       *)
(* CONTAIN A NUMBER.  THESE COMMENTS ARE REFERENCES TO   *)
(* NOTES ON THE USE OF PASCAL DATA TYPES AND STRUCTURES.*)
```

```
(* THE NOTES APPEAR AT THE END OF THE PROCEDURE.        *)
(*                                                      *)
(*------------------------------------------------------*)
(* THIS PROCEDURE TAKES A JULIAN DATE AND CONVERTS IT   *)
(* TO "DAY, MONTH, YEAR" FORMAT BY CALLING THE MPE      *)
(* INTRINSIC FMTCALENDAR.                               *)
(*------------------------------------------------------*)
PROGRAM PASPROC(INPUT,OUTPUT);

TYPE SMALLINT  = -32768..32767;
     DBBUF_REC = RECORD CASE INTEGER OF                  (* 1 *)
                 0 : (DBBUF_CHAR : PACKED ARRAY [1..4102] OF CHAR);
                 1 : (DBBUF_INT  : ARRAY [1..2051] OF SMALLINT);
                 END;
     BASEARRAY = ARRAY [1..130]  OF SMALLINT;
     LISTARRAY = ARRAY [1..53]   OF SMALLINT;
     REGARRAY  = ARRAY [1..150]  OF SMALLINT;
     STORARRAY = ARRAY [1..64]   OF SMALLINT;
     LINEARRAY = RECORD                                  (* 2 *)
                    BLINEBUF : PACKED ARRAY [1..138]  OF CHAR;
                 END;
     DBARRAY   = RECORD
                    BDB_NAME : PACKED ARRAY [1..26]   OF CHAR;
                 END;
     DSARRAY   = RECORD
                    BDS_NAME : PACKED ARRAY [1..16]    OF CHAR;
                 END;

                                                         (* 3 *)
PROCEDURE MPROC (VAR CALLAGAIN, PAGE_EJECT:        SMALLINT;
                 VAR BASE_IDS:                     BASEARRAY;
                 VAR DBBUF:                        DBBUF_REC;
                 VAR DSET_LIST, DATA_BASE_LIST:    LISTARRAY;
                 VAR LINEBUF:                      LINEARRAY;
                 VAR REGARR:                       REGARRAY;
                 VAR USERSTOR:                     STORARRAY;
                 VAR USERPARAM:                    SMALLINT;
                 VAR DATA_BASE_NAME:               DBARRAY;
                 VAR DATA_SET_NAME:                DSARRAY;
                 VAR CALL_NUM, LINES_LEFT, PAGENO: SMALLINT;
                 VAR NUM_DSETS, NUM_DBASES:        SMALLINT);



TYPE DAY_REC  = RECORD CASE INTEGER OF
                0 : (D_DAY : INTEGER);
                1 : (DAY   : ARRAY [1..2] OF SMALLINT);
                END;
     YEAR_REC = RECORD CASE INTEGER OF
                0 : (D_YEAR : INTEGER);
                1 : (YEAR   : ARRAY [1..2] OF SMALLINT);
                END;
     JDATE_REC = RECORD CASE INTEGER OF
                 0 : (DAY_YR : PACKED RECORD
                                  YEAR : 0..127;
                                  DAY  : 0..511;
                               END);
                 1 : (DATE : SMALLINT);
                 END;

VAR FMTDATE : PACKED ARRAY [1..17] OF CHAR;
    DAY     : DAY_REC;
    YEAR    : YEAR_REC;
    HOLD    : INTEGER;
    JDATE   : JDATE_REC;
    M, N    : SMALLINT;
```

```
    HOLD_BUF: RECORD CASE INTEGER OF
                0 : (SINGLE_INT : ARRAY [1..2] OF SMALLINT);
                1 : (DOUBLE_INT : INTEGER);
              END;

PROCEDURE FMTCALENDAR;  INTRINSIC;

(*-------------------------------------------------------*)
(* THERE ARE TWO DATES IN THE DATA BASE TO BE FORMATTED.  *)
(* ONE IS IN THE ORDER-D DATA SET, THE OTHER IS IN THE    *)
(* SHIP-D DATA SET.  BECAUSE THE SETS HAVE BEEN JOINED,   *)
(* IT IS POSSIBLE TO GET BOTH DATES AND FORMAT THEM.      *)
(* NOTE: IT IS THE APPLICATION'S RESPONSIBILITY TO KNOW    *)
(* WHERE IN THE DATA BUFFER THE VALUE IS LOCATED.         *)
(*-------------------------------------------------------*)
(*-------------------------------------------------------*)
(* THE DATE IS IN A TWO WORD INTEGER AND NEEDS TO BE RE-  *)
(* FORMATTED BEFORE PASSING TO FMTCALENDAR.               *)
(*-------------------------------------------------------*)
(*-------------------------------------------------------*)
(* IF CALL_NUM IS ONE THEN WE CAN ASSUME THE ENTRY IS     *)
(* FROM THE ORDER-D DATA SET, SINCE IT IS THE FIRST DATA  *)
(* SET MENTIONED IN THE JOIN COMMAND.  IF CALL_NUM IS NOT *)
(* ONE THEN WE CAN ASSUME THAT THE PROCEDURE HAS BEEN     *)
(* CALLED AGAIN, AND WE NOW HAVE THE ENTRY FROM THE       *)
(* SHIP-D DATA SET.                                       *)
(*-------------------------------------------------------*)

BEGIN

IF CALL_NUM =1 THEN                          (* FIRST DATE *)
   BEGIN
   HOLD_BUF.SINGLE_INT[1] := DBBUF.DBBUF_INT[4];
   HOLD_BUF.SINGLE_INT[2] := DBBUF.DBBUF_INT[5];
   END
ELSE                                         (* SECOND DATE *)
   BEGIN
   HOLD_BUF.SINGLE_INT[1] := DBBUF.DBBUF_INT[6];
   HOLD_BUF.SINGLE_INT[2] := DBBUF.DBBUF_INT[7];
   END;

YEAR.D_YEAR := HOLD_BUF.DOUBLE_INT DIV 1000;       (* ISOLATE YEAR *)
HOLD := YEAR.D_YEAR * 1000;
DAY.D_DAY := HOLD_BUF.DOUBLE_INT - HOLD;
JDATE.DAY_YR.DAY := DAY.DAY[2];             (* DAY IN BITS 0-6 *)
JDATE.DAY_YR.YEAR := YEAR.YEAR[2];         (* YEAR IN BITS 7-15 *)
FMTCALENDAR (JDATE.DATE, FMTDATE);

(*-------------------------------------------------------*)
(* PUT THE FORMATTED DATE INTO THE OUTPUT BUFFER USED BY  *)
(* QUERY FOR THE REPORT OUTPUT LINE.  AGAIN, CALL_NUM IS  *)
(* USED TO DETERMINE WHERE IN THE OUTPUT BUFFER TO PLACE  *)
(* THE VALUE.  ALSO, IF CALL_NUM IS ONE THEN WE NEED TO   *)
(* SET THE DATA BASE NAME, THE DATA SET NAME, AND THE     *)
(* CALLAGAIN PARAMETERS IN ORDER TO GET THE ENTRY FROM    *)
(* THE SHIP-D DATA SET TO FORMAT THE SHIPPING DATE.       *)
(*-------------------------------------------------------*)

IF CALL_NUM = 1 THEN               (* ORDER-D DATA SET *)
   BEGIN
   M := 1;
   FOR N := 19 TO 35 DO
      BEGIN
      LINEBUF.BLINEBUF[N] := FMTDATE [M];
      M := M + 1;
      END;
```

```
   DATA_BASE_NAME.BDB_NAME := 'ACCTS        ';
   DATA_SET_NAME.BDS_NAME := 'SHIP-D  ';
   CALLAGAIN := -1;                                    (* 4 *)
   END

ELSE                                (* SHIP-D DATA SET *)
   BEGIN
   M := 1;
   FOR N := 39 TO 55 DO
      BEGIN
      LINEBUF.BLINEBUF[N] := FMTDATE [M];
      M := M + 1;
      END;
   CALLAGAIN := 0;
   END;

END;

BEGIN
END.
```

PASCAL Notes

(* 1 *)

Variant records can be used when it is necessary to access arrays that contain different data types. Variant records are allocated the maximum amount of storage needed for the record. Because of this it is possible to 'overlay' the fields so data can be accessed as different types. For example, DBBUF can now be accessed as character data if DBBUF_CHAR is referenced, or as a single word integer data if DBBUF_INT is referenced.

(* 2 *)

For proper storage (one character per byte), character data should be in a PACKED ARRAY OF CHAR (PACs). However, PACs have byte addresses and the data passed between QUERY and the user-defined procedure must have word addresses. When a PAC is declared as a field in a RECORD, it has a word address.

(* 3 *)

QUERY passes all parameters as word addresses. The parameters must be declared as VAR so that indirect addressing is performed, otherwise, the addresses will be taken as values.

(* 4 *)

When PASCAL assigns TRUE to a BOOLEAN, a one is placed in the left byte of the BOOLEAN value. When SPL tests TRUE or FALSE, it tests for odd or even respectively. The PASCAL TRUE will be considered FALSE. So CALLAGAIN is declared to be a SMALLINT and is assigned the SPL convention of minus one for TRUE and zero for FALSE.

Additional Comments

If you wish to use PASCAL I/O to help debug your procedure you must do it the following way:

```
REWRITE (output, '$STDLIST');
WRITELN (output, 'message');
```

FORTRAN - MPROC Procedure

```
$CONTROL FREE
 #-------------------------------------------------------#
 # YOU WILL NOT WANT TO INITIALIZE THE USL FILE IF YOU   #
 # ARE COMPILING DIFFERENT SUBPROGRAMS INTO IT.          #
 #-------------------------------------------------------#
 $CONTROL MAP, USLINIT, LOCATION
 $CONTROL SEGMENT=MULTISEG
 #-------------------------------------------------------#
 #                                                       #
 #                       MPROC                           #
 #                                                       #
 # EXAMPLE OF A FORTRAN USER-DEFINED PROCEDURE, USED     #
 # WITH A REPORT STATEMENT.                              #
 #                                                       #
 #-------------------------------------------------------#
 # THIS PROCEDURE TAKES A JULIAN DATE AND CONVERTS IT    #
 # TO "DAY, MONTH, YEAR" FORMAT BY CALLING THE MPE       #
 # INTRINSIC FMTCALENDAR.                                #
 #-------------------------------------------------------#
 #
 #-----------------------------------------#
 #          MPROC DECLARATION              #
 #-----------------------------------------#
 #
 SUBROUTINE MPROC (CALLAGAIN, PAGEEJECT, BASEIDS, DBBUF, &
                   DSETLIST, DBLIST, LINEBUF, REGARR, &
                   USERSTOR, USERPARAM, DBNAME, &
                   DSNAME, CALLNUM, LINESLEFT, PAGENO,  &
                   NUMDSETS, NUMDBASES)
 #
 LOGICAL CALLAGAIN, PAGEEJECT
 INTEGER BASEIDS(130), DBBUF(2051), DSETLIST(53), DBLIST(53)
 INTEGER LINEBUF(69), REGARR(150), USERSTOR(64)
 INTEGER USERPARAM
 INTEGER DBNAME(13), DSNAME(8)
 INTEGER CALLNUM, LINESLEFT, PAGENO, NUMDSETS, NUMDBASES
 #
 #
 INTEGER FMTDATE (9)
 CHARACTER BFMTDATE (18)
 EQUIVALENCE (FMTDATE,BFMTDATE)
 #
 INTEGER*4 DHOLD, DDAY, DYEAR, HOLDYR
 INTEGER DAY(2), YEAR(2), HOLD(2)
 EQUIVALENCE (DDAY,DAY), (DYEAR,YEAR), (DHOLD,HOLD)
 #

 INTEGER INTDBNAME(13), INTDSNAME(8)
 CHARACTER*26 CHDBNAME
 CHARACTER*16 CHDSNAME
 EQUIVALENCE (INTDBNAME,CHDBNAME), (INTDSNAME,CHDSNAME)
 #
 LOGICAL JDATE
 #
```

```
#
SYSTEM INTRINSIC FMTCALENDAR
#
#---------------------------------------------------------#
# THERE ARE TWO DATES IN THE DATA BASE TO BE FORMATTED.  #
# ONE IS IN THE ORDER-D DATA SET, THE OTHER IS IN THE    #
# SHIP-D DATA SET.  BECAUSE THE SETS HAVE BEEN JOINED,   #
# IT IS POSSIBLE TO GET BOTH DATES AND FORMAT THEM.      #
# NOTE: IT IS THE APPLICATION'S RESPONSIBILITY TO KNOW    #
# WHERE IN THE DATA BUFFER THE VALUE IS LOCATED.         #
#---------------------------------------------------------#
#---------------------------------------------------------#
# THE DATE IS IN A TWO WORD INTEGER AND NEEDS TO BE RE-  #
# FORMATTED BEFORE PASSING TO FMTCALENDAR.               #
#---------------------------------------------------------#
#---------------------------------------------------------#
# IF CALLNUM IS ONE THEN WE CAN ASSUME THE ENTRY IS      #
# FROM THE ORDER-D DATA SET, SINCE IT IS THE FIRST DATA  #
# SET MENTIONED IN THE JOIN COMMAND.  IF CALLNUM IS NOT  #
# ONE THEN WE CAN ASSUME THAT THE PROCEDURE HAS BEEN     #
# CALLED AGAIN, AND WE NOW HAVE THE ENTRY FROM THE       #
# SHIP-D DATA SET.                                       #
#---------------------------------------------------------#
#
IF (CALLNUM .GT. 1) GOTO 100
HOLD(1) = DBBUF(4)
HOLD(2) = DBBUF(5)
GOTO 200
#
100 HOLD(1) = DBBUF(6)
HOLD(2) = DBBUF(7)
#
200 DYEAR = DHOLD / 1000
HOLDYR = DYEAR * 1000
DDAY = DHOLD - HOLDYR
JDATE[0:7] = BOOL (YEAR(2))
JDATE[7:9] = BOOL (DAY(2))
#

DO 225 N = 1, 18
225 BFMTDATE(N) = " "
#
CALL FMTCALENDAR (JDATE, BFMTDATE)
#
#---------------------------------------------------------#
# PUT THE FORMATTED DATE INTO THE OUTPUT BUFFER USED BY  #
# QUERY FOR THE REPORT OUTPUT LINE.  AGAIN, CALL'NUM IS  #
# USED TO DETERMINE WHERE IN THE OUTPUT BUFFER TO PLACE  #
# THE VALUE.  ALSO, IF CALL'NUM IS ONE THEN WE NEED TO   #
# SET THE DATA BASE NAME, THE DATA SET NAME, AND THE     #
# CALLAGAIN PARAMETERS IN ORDER TO GET THE ENTRY FROM    #
# THE SHIP-D DATA SET TO FORMAT THE SHIPPING DATE.       #
#---------------------------------------------------------#
#
IF (CALLNUM .GT. 1) GOTO 300
M = 1
DO 250 N = 10, 18
    LINEBUF(N) = FMTDATE(M)
250 M = M + 1
CHDBNAME = "ACCTS                    "
CHDSNAME = "SHIP-D          "
M = 1
DO 275 N = 1, 13
    DBNAME(N) = INTDBNAME(M)
275 M = M + 1
M = 1
```

```
DO 285 N = 1, 8
    DSNAME(N) = INTDSNAME(M)
285 M = M + 1
CALLAGAIN = .TRUE.
GOTO 400
#
300 M = 1
DO 350 N = 21, 29
    LINEBUF(N) = FMTDATE(M)
350 M = M + 1
CALLAGAIN = .FALSE.
#
400 RETURN
END
```

QPROC Procedure

The following examples show the QPROC user-defined procedure in two languages: SPL and COBOL. This procedure displays the parameter values when it is called from QUERY. You can interactively change the parameters which can be modified by user-defined procedures. The changes can affect the data returned. The procedure can be terminated before all the report entries have been processed by entering (CONTROL) Y at an input prompt. The procedure will terminate after that sequence of parameter prompts is finished and the report command will terminate.

SPL - QPROC Procedure

```
$PAGE "QPROC: Program Description."
$CONTROL MAP, USLINIT, SUBPROGRAM
$CONTROL SEGMENT=QPROC
<<*****************************************************************>>
<<                                                                 >>
<<                          Q P R O C                              >>
<<                                                                 >>
<<          SPL Example of QUERY user callable procedure           >>
<<                                                                 >>
<<*****************************************************************>>

begin

INTRINSIC     PRINT,
              ASCII,
              DASCII,
              DEBUG,
              READ,
              BINARY;



PROCEDURE QPROC(CALLAGAIN,PAGE'EJECT,BASE'IDS,DBBUF, DSET'LIST,
                DATA'BASE'LIST, LINEBUF, REGARR, USERSTOR, USERPARAM,
                DATA'BASE'NAME, DATA'SET'NAME, CALL'NUM, LINES'LEFT,
                PAGENO, NUM'DSETS, NUM'DBASES );

LOGICAL         CALLAGAIN, PAGE'EJECT;
INTEGER ARRAY   BASE'IDS, DBBUF, DSET'LIST, DATA'BASE'LIST,
                LINEBUF, REGARR, USERSTOR, USERPARAM,
                DATA'BASE'NAME, DATA'SET'NAME;
INTEGER         CALL'NUM, LINES'LEFT, PAGENO, NUM'DSETS,
                NUM'DBASES;
```

```
$PAGE "QPROC: Procedure Global Variables."
begin


<<-------------------------------------------------------------->>
<<                       LOCAL DECLARATIONS                     >>
<<-------------------------------------------------------------->>


          <<---------------------------------------------->>
          <<-------- REDEFINE SOME PASSED PARAMETERS ------->>
          <<---------------------------------------------->>

BYTE ARRAY       BASE'IDS'B(*)=BASE'IDS;
BYTE ARRAY       DBBUF'B(*)=DBBUF;

          <<---------------------------------------------->>
          <<--------------- NEW LOCAL STORAGE -------------->>
          <<---------------------------------------------->>

INTEGER          NUM'CHAR;
INTEGER          I, J, K, L;  << COUNTERS For DO-WHILE / DO-until >>
INTEGER          REG'INDEX;

INTEGER          END'PTR;
INTEGER          START'PTR;
INTEGER          DEC'PTR;     << DECIMAL POINT >>

ARRAY            BUF(0:50 );
BYTE ARRAY       BUF'B(*) = BUF;
INTEGER          BUF'INDX;

ARRAY            BUF1(0:50 );
 BYTE ARRAY      BUF1'B(*) = BUF1;

LONG ARRAY       TEMP'L(0:1 );
 INTEGER ARRAY   TEMP'I(*) = TEMP'L;

$PAGE "QPROC: Outer block."


<<**************************************************************>>
<<                       START OF QPROC                         >>
<<**************************************************************>>


PRINT( BUF,   0, %60 );
PRINT( BUF,   0, %60 );
move BUF'B(0): = "            ********    PROCEDURE QPROC   *******";
PRINT( BUF, -49, %40 );


<<-------------------------------------------------------------->>
<<                      CALL AGAIN                               >>
<<-------------------------------------------------------------->>

move BUF : = "-- CALL AGAIN -- (CHANGE to True or False)";
PRINT( BUF,   0, %40 );
PRINT( BUF, -42, %40 );
if CALLAGAIN = TRUE  then move BUF := "--TRUE-- ";
if CALLAGAIN = FALSE then move BUF := "--FALSE--";
PRINT( BUF,  -9, %40 );
I := READ( BUF,  -5 );
if I<>0 and (BUF'B = "T" or BUF'B = "t") then CALLAGAIN := TRUE;
if I<>0 and (BUF'B = "F" or BUF'B = "f") then CALLAGAIN := FALSE;
```

```
<<-------------------------------------------------------------->>
<<                       PAGE EJECT                             >>
<<-------------------------------------------------------------->>
move BUF := "-- PAGE EJECT -- (CHANGE to True or False)";
PRINT( BUF,   0, %40 );
PRINT( BUF, -42, %40 );
if PAGE'EJECT = TRUE then move BUF := "--TRUE-- ";
if PAGE'EJECT = FALSE then move BUF := "--FALSE--";
PRINT( BUF, -9, %40 );
I := READ( BUF, -5 );
if I<>0 and (BUF'B = "T" or BUF'B = "t") then PAGE'EJECT:= TRUE;
if I<>0 and (BUF'B = "F" or BUF'B = "f") then PAGE'EJECT:= FALSE;

<<-------------------------------------------------------------->>
<<                       BASE ID'S                              >>
<<-------------------------------------------------------------->>
move BUF := "-- BASE ID'S --";
PRINT( BUF,   0, %40 );
PRINT( BUF, -15, %40 );
move BUF'B(0) := (79(" ") );
if BASE'IDS(1) <> %020040  then begin
  NUM'CHAR := ASCII( BASE'IDS(0), 10, BUF'B );
  move BUF'B(NUM'CHAR) := BASE'IDS'B(2), (24 );
  if BASE'IDS(14) <> %020040 then begin
    NUM'CHAR := ASCII( BASE'IDS(13), 10, BUF'B(35) );
    move BUF'B(NUM'CHAR+35) := BASE'IDS'B(28), (24 );
    end;
  end;
PRINT( BUF, -79, %40 );

$PAGE
<<-------------------------------------------------------------->>
<<                    DATA BASE RECORD BUFFER                   >>
<<-------------------------------------------------------------->>
move BUF := "-- DB BUFFER -- ";
PRINT( BUF,   0, %40 );

PRINT( BUF, -15, %40 );
move BUF'B(0) := (79(" ") );
For I := 0 until 10 do
  ASCII( DBBUF(I), 8, BUF'B(I*7) );
PRINT (BUF, -79, %40 );
move BUF'B(0) := (79(" ") );
For I := 0 step 7 until 70 do begin
  BUF'B(I+1) := DBBUF'B(I/7*2 );
  if BUF'B(I+1)<%40 or BUF'B(I+1)>%176 then BUF'B(I+1):=%40;
  BUF'B(I+4) := DBBUF'B(I/7*2+1 );
  if BUF'B(I+4)<%40 or BUF'B(I+1)>%176 then BUF'B(I+4):=%40; end;
PRINT (BUF, -79, %40 );

<<-------------------------------------------------------------->>
<<                  LIST OF DATA SETS                           >>
<<-------------------------------------------------------------->>
move BUF := "-- DATA SET LIST --";
PRINT( BUF,   0, %40 );
PRINT( BUF, -19, %40 );
move BUF'B(0) := (79(" ") ); J := 0;
For I := 0 step 8 until 72 do begin
  NUM'CHAR := ASCII( DSET'LIST(J), 10, BUF'B(I) ); J := J+1; end;
PRINT (BUF, -79, %40 );

<<-------------------------------------------------------------->>
<<                    LIST OF DATA BASES                        >>
<<-------------------------------------------------------------->>
move BUF := "-- DATA BASE LIST --";
```

```
PRINT( BUF,   0, %40 );
PRINT( BUF, -20, %40 );
move BUF'B(0) := (79(" ") ); J := 0;
For I := 0 step 8 until 72 do begin
  NUM'CHAR := ASCII( DATA'BASE'LIST(J), 10, BUF'B(I) );J:=J+1;end;
PRINT (BUF, -79, %40 );

<<-------------------------------------------------------------->>
<<                        PRINT LINE BUFFER                     >>
<<-------------------------------------------------------------->>

move BUF'B:="--LINE BUFFER -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -16, %40 );
PRINT( LINEBUF, -79, %40 );
I := READ( LINEBUF, -79 );
PRINT'REGARR:
$PAGE
<<-------------------------------------------------------------->>
<<                    ARRAY OF REGISTERS                        >>
<<             (assume real numbers in registers)               >>
<<-------------------------------------------------------------->>
move BUF := "-- REGISTER ARRAY -- ";

PRINT( BUF,   0, %40 );
PRINT( BUF, -20, %40 );
move BUF'B(0) := (79(" "));
J := 0;
For I := 0 step 13 until 78 do begin
  move TEMP'I := REGARR(J), (4 );
  NUM'CHAR:=DASCII( FIXR(REAL(TEMP'L)), 10, BUF'B(I) );
  J := J + 5;
  end;
PRINT( BUF, -79, %40 );
I := READ( BUF, -79 );
END'PTR := 0;
START'PTR := 0;
DEC'PTR := 0;
BUF'INDX := 0;
REG'INDEX := 0;
if J = 0 then go to SKIP'REGISTER'LOAD;      << SEE BELOW >>

          <<---------------------------->>
          <<------- REGISTER LOAD ------>>
          <<---------------------------->>

For J := 0 until I do
  begin
  if BUF'B(J) = "," then END'PTR := J;
  if BUF'B(J) = "," or J = I then
    begin
  if DEC'PTR = START'PTR then DEC'PTR := END'PTR; <<NO DECIMAL>>
    K := BINARY( BUF1'B, BUF'INDX );
    if <> then K := 0;
    TEMP'L := LONG(REAL(K) );
    For L:= 1 until (BUF'INDX-(DEC'PTR-START'PTR)) do
      begin
      TEMP'L := TEMP'L * .1L0;
      end;
    move REGARR(REG'INDEX) := TEMP'I, (4);
    REG'INDEX := REG'INDEX + 5;
    BUF'INDX := 0;
    DEC'PTR := J;
    end;
  if BUF'B(J) = "," then START'PTR := J+1;
  if BUF'B(J) = "," then DEC'PTR := J+1;
```

```
  if BUF'B(J) = "." then DEC'PTR := J;
  if BUF'B(J) <> "," and BUF'B(J) <> "." then
    begin
    move BUF1'B(BUF'INDX) := BUF'B(J), (1);
    BUF'INDX := BUF'INDX + 1;
    end;
  end;


SKIP'REGISTER'LOAD:                          << SKIP REGISTER LOAD >>

$PAGE
<<--------------------------------------------------------------->>
<<                      USER STORE ARRAY                         >>
<<--------------------------------------------------------------->>
move BUF := "-- USER STORE -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -16, %40 );
PRINT( USERSTOR, -79, %40 );
I := READ( USERSTOR, -79 );

<<--------------------------------------------------------------->>
<<                      USER PARAMETER                           >>
<<--------------------------------------------------------------->>
move BUF := "-- USER PARAMETER -- (INTEGER) ";
PRINT( BUF, -30, %40 );
NUM'CHAR := ASCII( USERPARAM, 10, BUF'B );
PRINT( BUF,   0, %40 );
PRINT( BUF, -NUM'CHAR, %40 );
GET'USERPARAM:
I := READ( BUF, -7 );
if I <> 0 then
   begin
  USERPARAM := BINARY( BUF'B, I );
  if <> then begin
    move BUF := "## DBINARY COULD NOT CONVERT ## ";
    PRINT( BUF, -31, %40 );
    go to GET'USERPARAM;
    end;
   end;

<<--------------------------------------------------------------->>
<<                       DATA BASE NAME                          >>
<<--------------------------------------------------------------->>
move BUF := "-- DATA BASE NAME -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -20, %40 );
PRINT( DATA'BASE'NAME, -26, %40 );
I := READ( DATA'BASE'NAME, -26 );


<<--------------------------------------------------------------->>
<<                      DATA SET NAME                            >>
<<--------------------------------------------------------------->>
move BUF := "-- DATA SET NAME -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -20, %40 );

PRINT( DATA'SET'NAME, -16, %40 );
I := READ( DATA'SET'NAME, -16 );

$PAGE
<<--------------------------------------------------------------->>
<<                      CALL NUMBER                              >>
<<--------------------------------------------------------------->>
move BUF := "-- CALL NUMBER -- ";
```

```
PRINT( BUF,   0, %40 );
PRINT( BUF, -17, %40 );
NUM'CHAR:=ASCII( CALL'NUM, 10, BUF'B );
PRINT( BUF, -NUM'CHAR, %40 );

<<---------------------------------------------------------------->>
<<                      LINES LEFT                                >>
<<---------------------------------------------------------------->>
move BUF := "-- LINES LEFT -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -16, %40 );
NUM'CHAR:=ASCII( LINES'LEFT, 10, BUF'B );
PRINT( BUF, -NUM'CHAR, %40 );

<<---------------------------------------------------------------->>
<<                      PAGE NUMBER                               >>
<<---------------------------------------------------------------->>
move BUF := "-- PAGE NUMBER -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -17, %40 );
NUM'CHAR:=ASCII( PAGENO, 10, BUF'B );
PRINT( BUF, -NUM'CHAR, %40 );

<<---------------------------------------------------------------->>
<<                    NUMBER OF DATA SETS                         >>
<<---------------------------------------------------------------->>
move BUF := "-- NUMBER OF DATA SETS -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -25, %40 );
NUM'CHAR:=ASCII( NUM'DSETS, 10, BUF'B );
PRINT( BUF, -NUM'CHAR, %40 );

<<---------------------------------------------------------------->>
<<                    NUMBER OF DATA BASES                        >>
<<---------------------------------------------------------------->>
move BUF := "-- NUMBER OF DATA BASES -- ";
PRINT( BUF,   0, %40 );
PRINT( BUF, -26, %60 );  << DOUBLE SPACE AFTER LAST LINE >>
NUM'CHAR:=ASCII( NUM'DBASES, 10, BUF'B );
PRINT( BUF, -NUM'CHAR, %40 );
end;
end.
```

COBOL - QPROC Procedure

```
$CONTROL USLINIT,SUBPROGRAM,DYNAMIC

 IDENTIFICATION DIVISION.
  PROGRAM-ID. QPROC.
  AUTHOR. HP.

      ****************************************************************
      *                                                              *
      *                       Q P R O C                              *
      *                                                              *
      *      COBOL Example of QUERY user callable procedure          *
      *                                                              *
      *--------------------------------------------------------------*
      *  QPROC requires the use of COBOLII's "Accept free" verb      *
      *  and should be compiled using a COBOLII compiler.  Numeric   *
      *  fields are displayed in zoned format for simplicity of      *
      *  the example.                                                *
      ****************************************************************

       ENVIRONMENT DIVISION.
```

```
 DATA DIVISION.

  WORKING-STORAGE SECTION.
  01  BUF.
      05 BUFX  OCCURS 136 TIMES PIC X.
  01  REG-NBR                  PIC S9(4) COMP.


  LINKAGE SECTION.
  01 CALLAGAIN             PIC S9(4)   COMP.
  01 PAGE-EJECT            PIC S9(4)   COMP.
  01 BASE-IDS.
     05 BASE-ID1           PIC S9(4)   COMP.
     05 BASE-NAME1         PIC X(24).
     05 BASE-ID2           PIC S9(4)   COMP.
     05 BASE-NAME2         PIC X(24).
     05 BASE-ID-OTHERS     PIC X(208).
  01 DBBUF.
     05 DBBUF-70          PIC X(70).
     05 DBBUFI REDEFINES DBBUF-70  OCCURS 35 TIMES
                          PIC S9(4) COMP.
     05 DBBUF-OTHER       PIC X(4026).
  01 DSET-LIST.
      05 DSET       OCCURS 100 TIMES PIC S9(4) COMP.
  01 DATA-BASE-LIST.
      05 DBASE      OCCURS 100 TIMES PIC S9(4) COMP.
  01 LINEBUF.
     05 LINEBUF-70         PIC X(70).
     05 LINEBUF-OTHER      PIC X(66).

  01 REGARR.
      05  REG   OCCURS 30 TIMES  PIC S9(18) COMP-3.
  01 USERSTOR.
     05 USERSTOR-70        PIC X(70).
     05 USERSTOR-OTHER     PIC X(58).
  01 USERPARAM             PIC S9(4)   COMP.
  01 DATA-BASE-NAME        PIC X(26).
  01 DATA-SET-NAME         PIC X(16).
  01 CALL-NUM              PIC S9(4)   COMP.
  01 LINES-LEFT            PIC S9(4)   COMP.
  01 PAGENO                PIC S9(4)   COMP.
  01 NUM-DSETS             PIC S9(4)   COMP.
  01 NUM-DBASES            PIC S9(4)   COMP.
$PAGE "QPROC: Procedure Division."

************************************************************
*                 PROCEDURE DIVISION                       *
************************************************************

 PROCEDURE DIVISION USING CALLAGAIN, PAGE-EJECT, BASE-IDS,
     DBBUF, DSET-LIST, DATA-BASE-LIST, LINEBUF, REGARR,
     USERSTOR, USERPARAM, DATA-BASE-NAME, DATA-SET-NAME,
     CALL-NUM, LINES-LEFT, PAGENO, NUM-DSETS, NUM-DBASES.

 BEGIN-QPROC.

 DISPLAY SPACE.
 DISPLAY SPACE.
 DISPLAY "      ******* PROCEDURE QPROC ******* "


*----------------------------------------------------------*
*                     CALL AGAIN                           *
*----------------------------------------------------------*
```

```
 DISPLAY SPACE.
 DISPLAY "-- CALL AGAIN -- (CHANGE to True or False) ".
 MOVE SPACE TO BUF.
 ACCEPT BUF.
 IF BUFX(1) = "T" OR BUFX(1) = "t" THEN MOVE -1 TO CALLAGAIN.
 IF BUFX(1) = "F" OR BUFX(1) = "f" THEN MOVE 0 TO CALLAGAIN.

*-----------------------------------------------------*
*                     PAGE EJECT                      *
*-----------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- PAGE EJECT -- (CHANGE to True or False) ".
 MOVE SPACE TO BUF.
 ACCEPT BUF.
 IF BUFX(1) = "T" OR BUFX(1) = "t" THEN MOVE -1 TO PAGE-EJECT.
 IF BUFX(1) = "F" OR BUFX(1) = "f" THEN MOVE 0 TO PAGE-EJECT.


*-----------------------------------------------------*
*                      BASE ID'S                      *
*-----------------------------------------------------*
 DISPLAY "-- BASE ID'S --".
 IF BASE-ID2 = 8224 THEN
   DISPLAY BASE-ID1, " ", BASE-NAME1
   ELSE
     DISPLAY BASE-ID1, " ", BASE-NAME1, "     ", BASE-ID2,
             " ", BASE-NAME2.

*-----------------------------------------------------*
*               DATA BASE RECORD BUFFER               *
*-----------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- DB BUFFER -- (Decimal and Character)".
 DISPLAY DBBUFI(1), " ", DBBUFI(2), " ", DBBUFI(3), " ",
         DBBUFI(4), " ", DBBUFI(5), " ", DBBUFI(6), " ",
         DBBUFI(7), " ", DBBUFI(8), " ", DBBUFI(9), " ",
         DBBUFI(10).
 DISPLAY DBBUF-70.


*-----------------------------------------------------*
*                  LIST OF DATA SETS                  *
*-----------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- DATA SET LIST--".
 DISPLAY DSET(1), " ", DSET(2), " ", DSET(3), " ",
         DSET(4), " ", DSET(5), " ", DSET(6), " ",
         DSET(7), " ", DSET(8), " ", DSET(9), " ".


*-----------------------------------------------------*
*                  LIST OF DATA BASES                 *
*-----------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- DATA BASE LIST--".
 DISPLAY DBASE(1), " ", DBASE(2), " ", DBASE(3), " ",
         DBASE(4), " ", DBASE(5), " ", DBASE(6), " ",
         DBASE(7), " ", DBASE(8), " ", DBASE(9), " ".

*-----------------------------------------------------*
*                PRINT LINE BUFFER                    *
*-----------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- LINE BUFFER --".
 DISPLAY LINEBUF-70.
 ACCEPT LINEBUF.
```

```
*-------------------------------------------------------*
*                 PRINT REGISTER ARRAY                  *
*               (Packed decimal is assumed)             *
*-------------------------------------------------------*

 DISPLAY SPACE.
 DISPLAY-REGISTERS.
    DISPLAY " ".
    DISPLAY "-- REGISTER ARRAY -- (0-8 Displayed)".
    DISPLAY REG(1), " ", REG(2), " ", REG(3), " ".
    DISPLAY REG(4), " ", REG(5), " ", REG(6), " ".
    DISPLAY REG(7), " ", REG(8), " ", REG(9), " ".

 ACCEPT-REGISTERS.
    DISPLAY "CHANGE A REGISTER (Yes/No)?".
    MOVE SPACE TO BUF.
    ACCEPT BUF.
    IF BUFX(1) = "Y" OR BUFX(1) = "y" THEN
       DISPLAY "ENTER REGISTER NUMBER (0-29)?"
       ACCEPT REG-NBR FREE
       IF REG-NBR > -1 AND REG-NBR < 31 THEN
          DISPLAY "ENTER VALUE FOR REGISTER"
          ADD 1 TO REG-NBR
          ACCEPT REG(REG-NBR) FREE.
    IF BUFX(1) = "Y" OR BUFX(1) = "y" THEN
       GO TO DISPLAY-REGISTERS.

*-------------------------------------------------------*
*                   USER STORE ARRAY                    *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- USER STORE --".
 DISPLAY USERSTOR-70.
 ACCEPT USERSTOR.

*-------------------------------------------------------*
*                   USER PARAMETER                      *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- USER PARAMETER --".
 DISPLAY USERPARAM.
 ACCEPT USERPARAM FREE.

*-------------------------------------------------------*
*                  DATA BASE NAME                       *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- DATA BASE NAME --".

 DISPLAY DATA-BASE-NAME.
 MOVE "  " TO DATA-BASE-NAME.
 ACCEPT DATA-BASE-NAME.

*-------------------------------------------------------*
*                   DATA SET NAME                       *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- DATA SET NAME --".
 DISPLAY DATA-SET-NAME.
 ACCEPT DATA-SET-NAME.

*-------------------------------------------------------*
*                    CALL NUMBER                        *
*-------------------------------------------------------*
```

```
 DISPLAY SPACE.
 DISPLAY "-- CALL NUMBER --".
 DISPLAY CALL-NUM.

*-------------------------------------------------------*
*                   LINES LEFT                          *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- LINES LEFT --".
 DISPLAY LINES-LEFT.

*-------------------------------------------------------*
*                   PAGE NUMBER                         *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- PAGE NUMBER --".
 DISPLAY PAGENO.

*-------------------------------------------------------*
*                 NUMBER OF DATA SETS                   *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- NUM-DSETS --".
 DISPLAY NUM-DSETS.

*-------------------------------------------------------*
*                 NUMBER OF DATA BASES                  *
*-------------------------------------------------------*
 DISPLAY SPACE.
 DISPLAY "-- NUMBER OF DATA BASES --".
 DISPLAY NUM-DBASES.
 GOBACK.
```

G

ASCII CHARACTER SET

This appendix is not available in this edition.