Data General

INFOS®
Query/Report Writer
User's Manual
(AOS)

093-000214-01

NOTICE

INFOS ®
Query/Report Writer
User's Manual
(AOS)
093-000214

Revision History:

Original Release- May 1979 (INFOS®Query Rev. 1.00)

First Revision - August 1980 (INFOS®Query Rev. 2.00)

This manual has been extensively revised from revision 00; therefore, change indicators have not been used.

Preface

This manual describes the Query and Report Writer facilities for Data General's INFOS® system. The Query enables you to select and display existing INFOS® records in an organized format, as opposed to how they look in the database. In addition, the Query allows you to retrieve groups of records without writing specific applications programs. The Report Writer enables you to output existing INFOS® records in the form of a report.

In the introduction that follows, we provide enough information about the INFOS® system so a person unfamiliar with it can easily understand the Query and Report Writer facilities. For additional information about the INFOS® system, consult the *INFOS® System User's Manual (AOS)* (93-000152).

For information about the Advanced Operating System (AOS), consult the *AOS Software Documentation Guide* (93-000202).

We have organized this manual to serve both novice users and experienced programmers. Throughout we use an INFOS® database designed and built for Caucus Car Parts as an example of a "typical" Query and Report Writer user. In Chapter 7, we show the Query and Report Writer facilities in use at Caucus. A copy of this database exists on the release tape for Query, which you should have; if not, check with your Data General Sales Representative. We suggest you use the tape to follow along with the examples as you read the manual. The best way to learn how to use the Query and the Report Writer is to actually use it. So, feel free to try out each example. The tape includes the Caucus Car Parts database and the various macros, record formats, and report formats they use. Note, however, in Chapter 7 we see how the people at Caucus define new formats and macros. These are not included on the tape; you can follow the steps outlined in Chapter 7 to create them yourself.

Chapter 1 introduces the general terms you'll need to understand before using the Query and the Report Writer. It also describes general rules you'll need to follow when using both facilities.

Chapter 2 introduces Caucus Car Parts and their database; you can use the tape while we explain the different Query and Report Writer commands.

Chapter 3 explains how you can tailor the Query and Report Writer to meet your data processing requirements. This chapter instructs you in creating record formats for use by the Query and report formats for use by the Report Writer. This chapter also tells you how to build macros of Query sessions.

Chapter 4 contains reference material for the Query. It explains how to convert certain types of data into a form the Query can use. We've written this chapter for someone with previous programming (preferably INFOS® system) experience.

Chapter 5 contains reference material for the Report Writer. It explains how to design report formats and how to use RWCHECK, the Stand-Alone Compiler.

Chapter 6 lists all the Query and Report Writer commands alphabetically. The pages of this chapter have color-keyed edges for easy reference. Each command, switch, and argument is fully explained, and there are more examples you can follow along with, using the Caucus database.

Chapter 7 describes the Query and Report Writer facilities at work at Caucus Car Parts. In the course of one day, the people at Caucus find many ways to use these facilities. "Watching" them at work, you'll see how these facilities can tackle everyday data processing problems. Any changes made to the database during this typical day, such as the creation of new macros, qformats, and/or new rformats, are not on the tape you received with the product. However, you should be able to create these "changes" as the people at Caucus create them.

Appendix A has a complete list of the Query/Report Writer facility's finite limits.

Appendix B lists all error messages you might possibly receive while working with the Query and Report Writer. An explanation of each message is included, in addition to advice on what to do if and when you receive the message.

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: {required$_1$ / required$_2$} which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering this argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
⤶	Press the NEW LINE or carriage return (CR) key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, in examples we use

THIS TYPEFACE TO SHOW YOUR ENTRY⤶
THIS TYPEFACE FOR SYSTEM QUERIES AND RESPONSES.

) is the AOS CLI prompt.

Contacting Data General

If you:

- Have comments on this manual -- Please use the prepaid Remarks Form that appears after the Index.
- Require additional manuals -- Please contact your local Data General sales representative.
- Experience software problems -- Please notify your local Data General systems engineer.

End of Preface

Contents

Chapter 1 - Introduction

Basic Concepts 1-1
Invoking the Query 1-4
Abbreviations 1-5
Command Arguments and Switches 1-5
/L Switch 1-6
Uppercase and Lowercase 1-6
Interrupting Activity on the Screen 1-7
Correcting Errors 1-7
Query Commands 1-7
Query Movement Commands 1-7
Selection and Display Commands 1-8
Other Commands 1-8
Summary 1-8

Chapter 2 - Learning to Use the Query / Report Writer

The "Typical" Application 2-1
The Individual Records 2-1
Qformats 2-1
Query Commands 2-2
Movement Commands 2-3
PATH command 2-4
KEY Command 2-4
UP and DOWN Commands 2-6
NEXT and PRIOR Commands 2-6
Selection and Display Commands 2-7
QFORMAT Command 2-7
RFORMAT Command 2-8
READ Command 2-10
CONDITION and SELECT Commands 2-13
CONDITION Command 2-13
SELECT Command 2-16
LEVELMARK Command 2-19
HELP Command 2-19
CLI Command 2-20
BYE Command 2-20
RWCHECK -- The Stand-Alone Compiler 2-21
Summary 2-21

Chapter 3 - Tailoring the Query

Qformats 3-1
Understanding Record Structures 3-2
Creating qformats 3-4
Using Your New Qformat 3-7
Rformats 3-7
Format Descriptor Lines 3-9
Comment Line 3-10
QFORMAT Line 3-10
LIN/PG and COL/LIN Lines 3-10
HEADER Lines 3-10
DEFINE Lines 3-11
DETAIL Lines 3-11
PICTURE Lines 3-11
SORT Lines 3-11
BREAK Lines 3-11
TOTAL Lines 3-11
General Rules for Defining Rformats 3-11
Using RWCHECK 3-12
Suggestions for Designing Reports 3-12
Using Rformats 3-13
Query Macros 3-13
Building Macros 3-13
Macro Building from the CLI 3-13
Saving Query Sessions 3-14
Tips for Building Macros 3-15
Self-contained Macros 3-15
Keep the Macro in the Query 3-16
Nesting Macros 3-16
Tying up Loose Ends 3-17
Suggestions for Using Macros 3-18
Summary 3-18

Chapter 4 - Query Reference Section: The .QFORMS File

Building the .QFORMS File 4-1
Syntax of the Query Format File 4-1
START_FORMAT Line 4-1
Field Descriptor Lines 4-2
END_FORMAT Line 4-2
Setting Up a .QFORMS File 4-2
Selection Procedures 4-2
Non-ASCII Formatting 4-4
Qformat Error Messages 4-6
Runtime Format Errors 4-6
Size and Scope of the .QFORMS File 4-6
Summary 4-7

Chapter 5 - Report-Writer Reference Section: The .RFORMS file

Building the .RFORMS File 5-1
Rformat Definition 5-1
START_REPORT Line 5-2
Comment Line 5-2
QFORMAT Line 5-2
Lines Per Page 5-2
Columns Per Line 5-3
HEADER Lines 5-3
DEFINE Lines 5-4
DETAIL Lines 5-4
PICTURE Lines 5-4
SORT Lines 5-5
BREAK Lines 5-5
TOTAL Lines 5-6
END_REPORT Line 5-7
Designing Reports 5-7
Size and Scope of the .RFORMS File 5-7
RWCHECK - The Stand-Alone Compiler 5-8
Using Rformats 5-11
Summary 5-11

Chapter 6 - Command Reference Section

QUERY 6-1
BYE 6-2
CLI 6-2
CONDITION 6-3
DOWN 6-5
HELP 6-6
KEY 6-8
LEVELMARK 6-9
NEXT 6-10
PATH 6-11
PRIOR 6-12
QFORMAT 6-13
READ 6-14
RFORMAT 6-17
SELECT 6-19
UP 6-21

Chapter 7 - A Day in the Life of Caucus Car Parts

Background 7-1

Appendix A - Query Function Limits

Appendix B - Error Messages

General Query Error Messages B-1
Query Movement Command Errors B-2
KEY Command B-2
PATH Command B-2
UP, DOWN, NEXT and PRIOR Commands B-2
Query Selection and Display Command Errors B-2
CONDITION Command B-2
READ and SELECT Commands B-3
RFORMAT Command B-3
QFORMAT Command B-4
Miscellaneous Query Command Errors B-4
CLI Command B-4
HELP Command B-4
LEVELMARK Command B-4
Qformat Syntax Errors B-4
Report Writer Runtime errors B-5
Rformat Syntax Errors B-5
General Rformat Syntax Errors B-5
BREAK Statement B-5
COLUMNS PER LINE Statement B-6
DEFINE Statement B-6
DETAIL Statement B-6
HEADER Statement B-6
LINES PER PAGE Statement B-7
PICTURE Statement B-7
QFORMAT Statement B-7
SORT Statement B-7
TOTAL Statement B-7

Tables

Table Caption

3-1 The Structure of a Typical Personnel Record 3-4
3-2 A List of Query Possibilities 3-5

4-1 Type Descriptions 4-5

5-1 Picture Characters 5-5

Illustrations

Figure Caption

1-1 The Route to Work Becomes Your Path 1-2
1-2 The Route to a Specific Car Part 1-3
1-3 Records, Fields, Qformats, Rformats 1-4

2-1 Index Structure for Caucus Car Parts 2-2
2-2 Contents of Caucus.QFORMS File and Two Sample Records 2-3
2-3 The PATH Command 2-4
2-4 The KEY/APPROX Command 2-5
2-5 The PATH and KEY Commands 2-5
2-6 Using the /DISPLAY Switch 2-9
2-7 The READ/TRAVERSE Command 2-10
2-8 Using the /RFORMAT Switch 2-12
2-9 The Query's Eyes 2-14
2-10 The SELECT/TRAVERSE Command 2-17
2-11 Using the SELECT/RFORMAT Command 2-18
2-12 The HELP Screen 2-19
2-13 The HELP OVERVIEW Screen 2-20

3-1 The .QFORMS File for Caucus Car Parts 3-2
3-2 The Internal Structure of Caucus Records 3-3
3-3 Full and Partial Record Formats 3-5
3-4 The .RFORMS File at Caucus Car Parts 3-8

4-1 Building Qformats 4-3

5-1 Sample Pages Produced by RWCHECK 5-10
5-2 STOCK VALUE Report 5-12
5-3 Report Created with the SELECT Command 5-13
5-4 An OLDIES Report Using a READ Command 5-14
5-5 An OLDIES Report Using a SELECT Command 5-15

6-A The Condition command syntax 6-3
6-B The HELP command 6-6

7-1 Caucus Car Parts' Query Set-up 7-1

Chapter 1
Introduction

Data General's INFOS® system Query facility is an interactive utility that enables you to work directly with existing INFOS files. By typing simple commands at your terminal, you can select and display individual INFOS records in fully readable formats. In addition, Query includes a Report Writer feature, an easy-to-use utility that enables you to produce a wide variety of reports from the same INFOS files.

At the heart of the Query is a powerful retrieval system that allows access to groups of records according to selection criteria that you choose. The Query adds an important dimension to the record selection process. Both experienced and inexperienced personnel can use the Query to select and display either the individual record or the group of records that they wish to see. In other words, you can often use the Query to do what formerly took the know-how of an applications programmer. Using a few commands, you can also immediately access your data and assemble it into a variety of report formats. In these ways, the INFOS Query facility helps lighten your applications programming load.

Throughout this manual, we refer to an INFOS database designed and built for Caucus Car Parts. (As explained in the Preface, you have a copy of this database on tape. We urge you to *play* with this database as we explain how Query works.) We'll use the Caucus database as an example while explaining the few basic terms and concepts you should understand before using the Query on your own.

Basic Concepts

As you may already know, the INFOS system is Data General's database-oriented file management system. A *database* is an organized collection of data *records*. Each record contains specific, stored information broken into *fields*. The purpose of Query is to let you access this stored data.

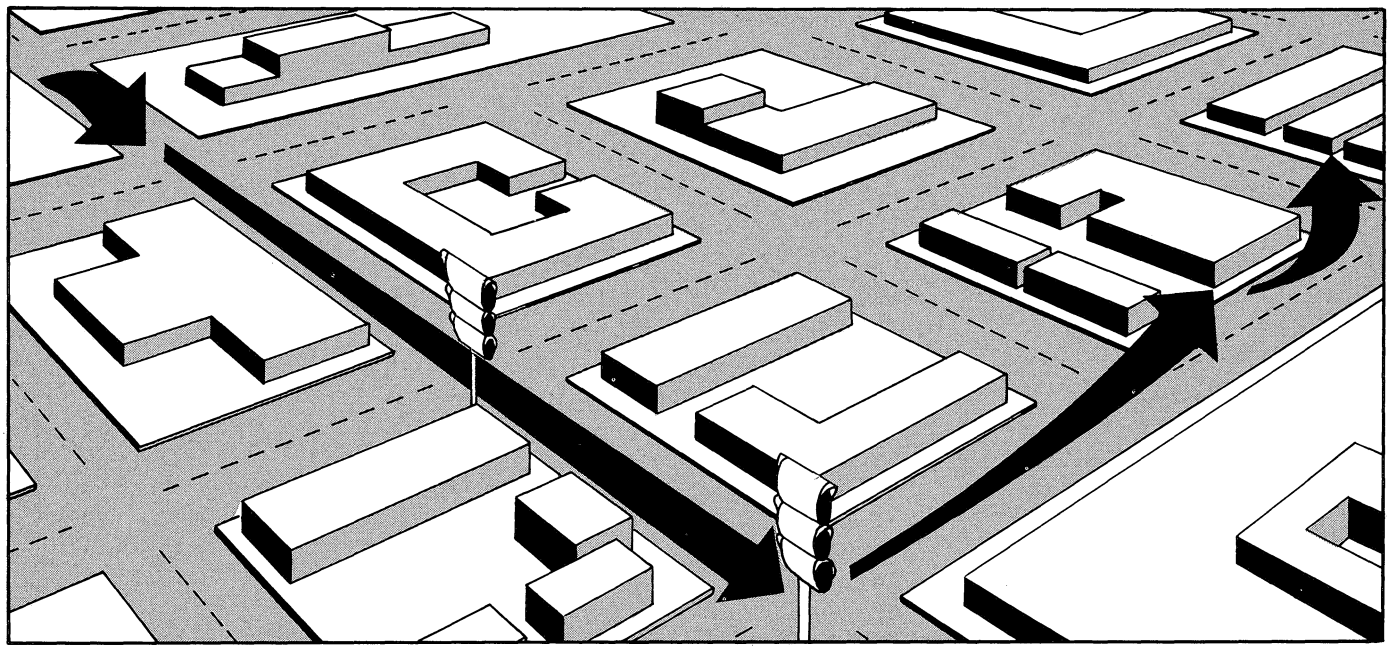
At the core of the INFOS system is the *index* structure which resembles an inverted tree; it branches down and out, providing pointers into the database.

At the top level of the structure you have one or more indexes. An index is a group of entries on the same level of the tree. Each entry in an index has a name or *key* associated with it.

An entry can point to a data record and/or another index. If the entry points to another index, that index is called a *subindex*. Subindexes further differentiate the organization of the records.

You use the index structure to move, one key at a time, down into the database. The route you follow to travel through the database is called your *path*. You can reach any point in the tree with a path, using keys (which include each index and subindex) as reference points.

In other words, a path through the database is similar to the route you might follow to reach a particular destination. You use the signposts (or keys) you pass along the route to name your path. As you can see in Figure 1-1, the path from your house to work might be called CORNER:SECOND STOP LIGHT: SUPERMARKET:OFFICE.



SD-02265

Figure 1-1. The Route to Work Becomes Your Path

Caucus Car Parts is a major distributor of American and imported car parts and accessories. The information stored in their database concerns the parts they sell. For instance, one index is called PARTS BY TYPE. To reach the data record for each car part, you'd start at the top of the index structure, at PARTS BY TYPE, and move down. The first key you "grab" onto would be the next index down, the selector subindex. This subindex contains keys named for the various car parts; i.e., AC UNIT (air conditioning unit), BATTERY, MUF (muffler), and so on. Once you find the key for the car part, then you need its part number. Each part type may have more one part number. For instance, the air conditioning unit for a Volvo would have a different part number than the one for a Renault. The part number is the next key down. In the Caucus database, the part number is the last key. Once you've reached the last key, you can easily "reach" into the database and pull out a record associated with a specific car part.

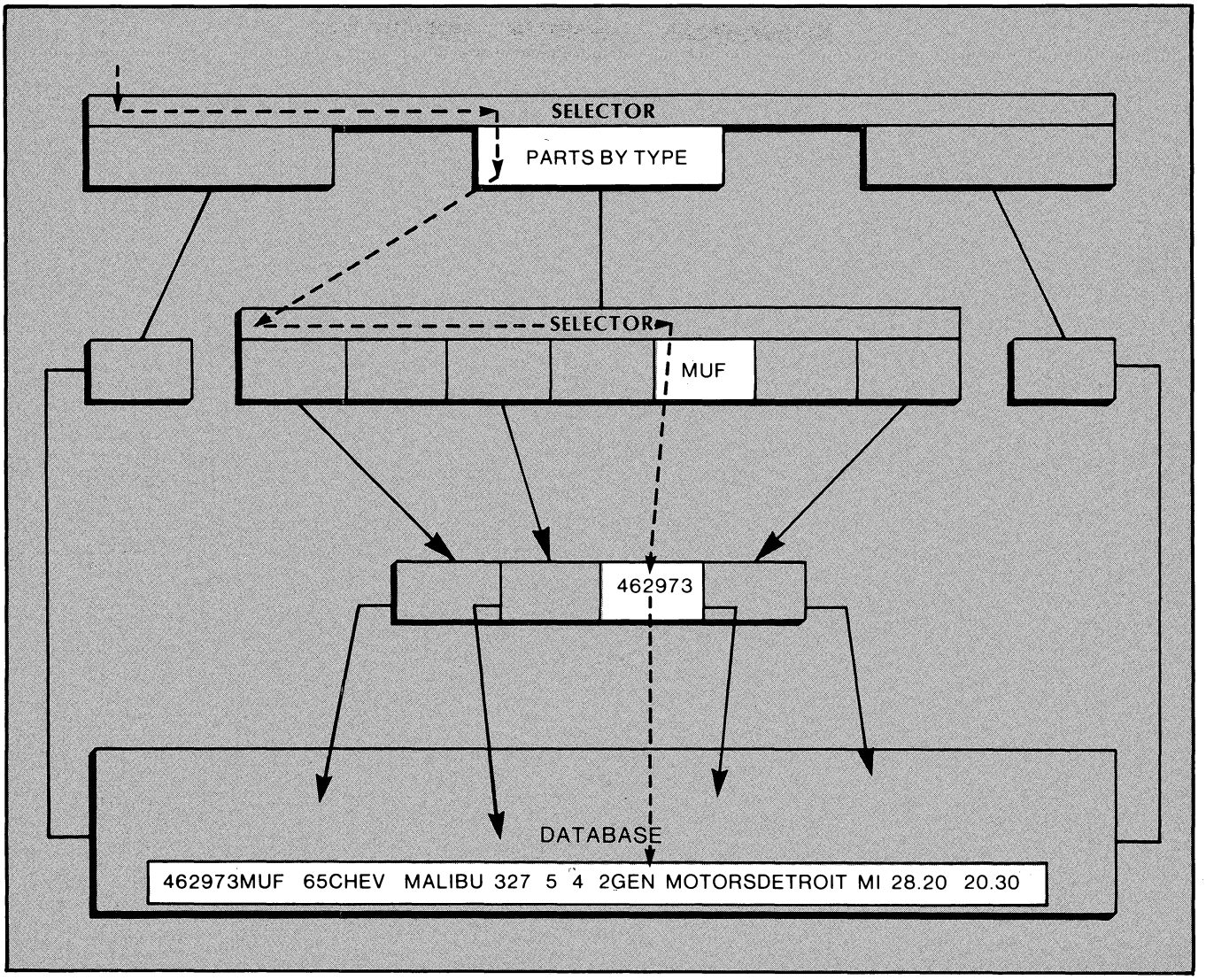
Figure 1-2 shows this concept.

Each record can have any number of fields, depending on the information you need to store. Caucus has two record types; one type contains information about the parts, the other contains information about purchase orders.

All the information contained in one record type is not necessarily needed for every application. Sometimes, you're only looking for the price, which is just one field in the PARTS record type. Query allows you to specify a *record format* that enables you to access only those particular fields you want to see. We call record formats *qformats* because the Query uses them for accessing information.

For ordering purposes, Caucus has a qformat called OPEN PO's. It uses certain fields from the PO (Purchase Order) record type. It allows you to select only the data associated with unfilled purchase orders, such as the PO number, the supplier's name and address, and the date the purchase order was sent out.

In addition, if you require specific information for a presentation, you could request that the Report Writer feature of the Query prepare it in a particular *report format*. To differentiate from qformats, report formats are called *rformats*. By declaring an rformat when you request information, you're asking the Query to find the information you need, and to use the Report Writer feature to display it in a readable, organized manner, complete with whatever column heads and computations you want included. For instance, you could use the Report Writer feature to print the information from the PO record type on Purchase Order forms.



SD-01336A

Figure 1-2. The Route to a Specific Car Part

Another aspect of the Report Writer feature is RWCHECK, the Stand-Alone Compiler. You use RWCHECK to verify the syntax of your rformats and to produce a sample page of your report. Although you need not be running INFOS to use RWCHECK or even have an INFOS file available, you must have already defined at least one qformat. We'll talk more about this later.

The Query enables you to scan through large indexes and retrieve database records that meet your established criteria. By declaring certain *conditions* for record retrieval, you use the Query to search for the information you need. For instance, if you wanted to know how many purchase orders for carburators Caucus Car Parts placed on June 12, 1979, you might specify the conditions: description = carburator, and date = 61279. When the Query searches through the PO index, it will retrieve only those purchase order records that meet your declared conditions. If you have the right rformat, the Query will present those purchase order records organized in the form of a report.

Figure 1-3 illustrates the concepts of *record, field, qformat, rformat.*

As you'll see, the opportunities for database inquiries are virtually limitless. Once you have established certain Query sequences, you can store them in a separate Query *macro* file for reuse later. A Query *macro* is simply a Query session you can access at a later time. Query macros are worksavers; they enable even inexperienced personnel to use the features of the system.

We have thus far introduced the basic terms and concepts that you need to understand how the Query and Report Writer fits into the INFOS system. In the next sections we'll talk about the different commands associated with the Query and Report Writer, and some general points you should keep in mind. In later chapters, we'll demonstrate, in detail, how you can best tailor the Query facilities to suit your data retrieval needs.

Before we talk about the specific commands, you should keep in mind a few general points. We may use specific commands as examples. Although we haven't explained the commands yet, it is not necessary to know about them to understand the general rules we're discussing. You don't have to be a programmer or a linquist to talk to the Query. The commands are very straightforward and English oriented, as you'll see.

Invoking the Query

Before you can use the Query facility to move through your INFOS database, you must "log on" to your system. Press the NEW LINE key on your keyboard and the system will display:

AOS xx.xx/EXEC x.xx date time

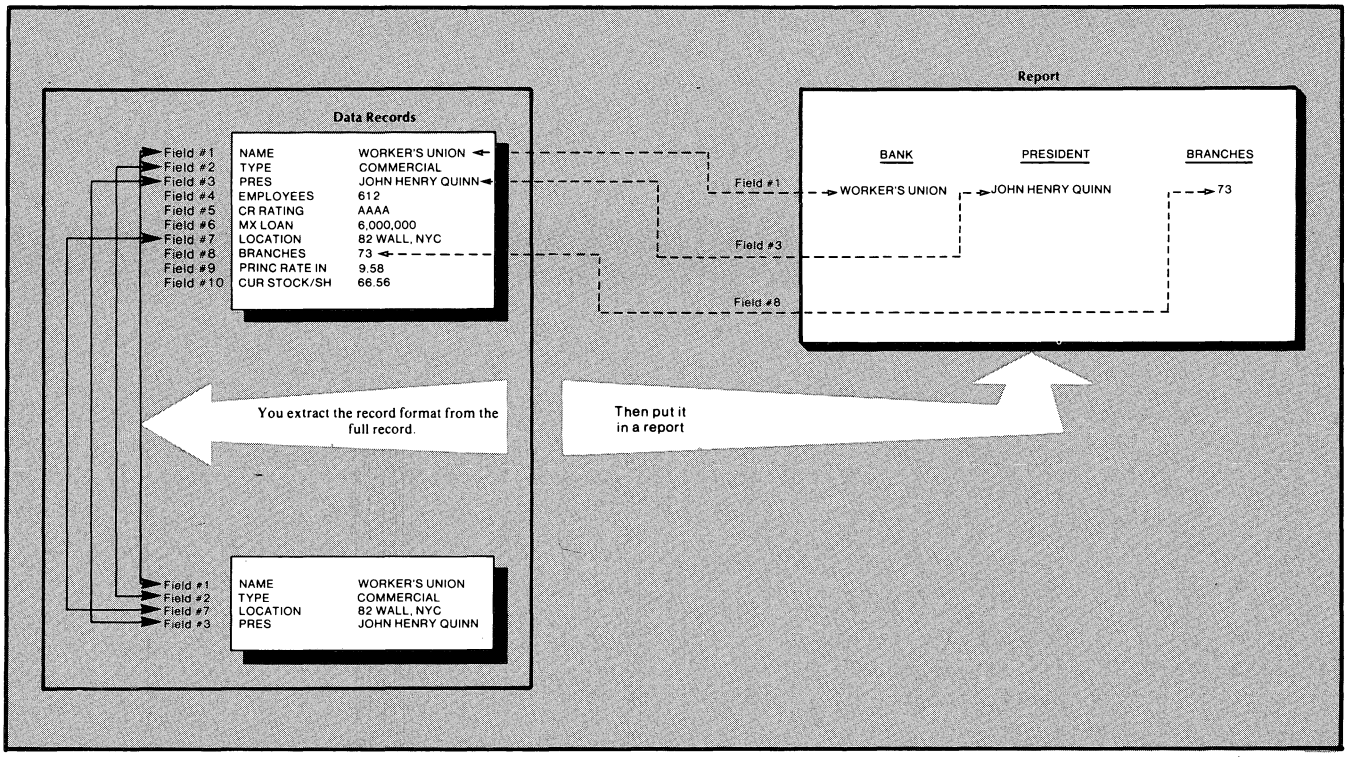
USERNAME:

PASSWORD:

You must now type in your USERNAME and PASSWORD, each followed by NEW LINE. If you have correctly logged on, your terminal will display

AOS CLI REV xx.xx date

)



SD-01334

Figure 1-3. Records, Fields, Qformats, Rformats

The right parenthesis,), is the AOS Command Line Interpreter (CLI) prompt. (Consult the *AOS Software Documentation Guide,* mentioned in the Preface, for more details on AOS and CLI.) Its presence indicates that your system is up and running. To activate the Query, type

```
)QUERY filename↵
```

where filename is the name of the INFOS index you want to use. If you type only

```
)QUERY↵
```

your system will respond with

INFOS INDEX NAME:

You must then supply the name of the INFOS index you want to access. For example, at Caucus you would type

```
)QUERY↵
INFOS INDEX NAME: CAUCUS↵
```

Your system will then return the message

INFOS Query / Report Writer .... At Your Service
>>

A pair of right angle brackets, >>, is the Query prompt. Its appearance means that the INFOS system has opened your file and that the Query is ready to accept commands.

Check with your system manager that INFOS is up and running if you get the following message:

INFOS PROCESS NOT RUNNING
)

Abbreviations

To save time when typing at the terminal, you can abbreviate entries.

To abbreviate a Query entry, simply type the minimum number of characters necessary to uniquely identify the entry. In other words, you can type the entry's shortest form, as long as it cannot be confused with another Query command, switch, filename, macro, or key (just like abbreviations in the CLI). For example, you could use N to abbreviate NEXT. However, if the abbreviation is nonunique, the Query will respond with

Command abbreviation not unique

You can abbreviate most of the Query's commands and switches. Except for PATH, PRIOR, CLI, CONDITION, READ, and RFORMAT, you can shorten all the Query's commands to the first letter. For example, you can shorten DOWN/SUBINDEXES to D/S:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> D/S↵
Path is PARTS BY TYPE:AC UNT
*** Subindex beneath this entry
```

You can also abbreviate the names of qformats and rformats. However, if more than one format starts with the same name, the first one defined will be the one you get. (As you'll see later, all qformats and rformats reside in special files.) If, for instance, you first created an rformat called STOCK VALUE and then called another STOCK, the Query will always give you STOCK VALUE when you specify STOCK because STOCK VALUE preceeds STOCK. Keep this in mind when you name qformats and rformats.

Command Arguments and Switches

Many commands take arguments directing the Query to perform the command more than once. An argument can either be a number between 1 and 32767 or, in some cases, the word ALL. In macro commands, however, the argument will be a filename. You must always leave a space between a command and its argument. If you don't, the Query will respond

Huh ? - That's not a Query command or a Query macro

For example:

```
>> NEXT3↵
Huh ? - That's not a Query command or a Query macro
>> NEXT 3↵
      AIR FT
      ALT
Path is PARTS BY TYPE:BATTRY
```

You can use various switches with several Query commands. A switch functions as a modifier to alter a command's original purpose. Always separate a switch from a command with a slash (/), but do not leave spaces between a command and its switch. If you separate the two entries with a space, the Query will interpret the switch as an argument.

For example:

```
>> Next /subindexes↵
Argument must be an integer between 1 and 32767
>> Next/subindexes↵
PATH is PO
*** Subindex beneath this entry
```

To execute a command, you must first receive the Query prompt > > , then type the command, and finally press the NEW LINE key. You may enter only one command sequence on a line. A command sequence consists of a Query command, a switch, and/or an argument. If you type in more than one command, the Query will respond

Argument must be an integer between 1 and 32767

When you use both arguments and switches in the same command sequence, you must enter the switch(es) first, followed by a space and then an argument. Thus, the basic format of a Query command follows the CLI (Command Line Interpreter) command syntax:

> > command < switch(es) > < argument >

In some commands, such as CONDITION and those used to build rformats, you must include a string literal. One note concerning this: if you include the string in quotation marks, make sure they match on both ends of the string. In other words, if you use double quotes (") on one end of the string, use double quotes on the other. Or, if you use a single quote ('), be sure to use it on both ends. In Chapter 3, when we discuss defining rformats, we'll talk more about this.

/L Switch

As we've said before, and we'll say again, the Query is an interactive facility used to select and display data from your INFOS database files. Most of your work will be done at the terminal. However, once you've chosen the records you want to see, you may want to store them some place. After all, a terminal screen display can't be put in your pocket and taken to a meeting. Most Query commands simply display data on the screen. To store the data in a file, which you in turn can print on a line printer, use either the /L or /L=filename switches.

You can use these switches with the HELP, QFORMAT, READ, RFORMAT, and SELECT commands.

If you use the /L=filename or /L switch with these commands, the Query sends the terminal display to a file: either the named file or the current list file, depending on which switch you use. You may access the file any time from the CLI. Note, however, you can not access either file from the Query with the CLI command. You must first end your Query session, return to the CLI, and then access these files. These files are not closed until you finish your Query session, thus making them inaccessible until then. However, if you use the /L switch again, directing the terminal display to a different list file, the system closes the first list file before opening the second.

You can send output to different files during a single Query session. However, you should remember that whichever list file was current when you entered Query will remain current throughout the entire Query session. This is the file you access each time you use the /L switch. If you want to send output to any other files, you use the /L=filename switch.

Anytime you use the /L or /L=filename switches, the Query will indicate how many records were sent to the text or list file. You'll receive either of the following messages on your console:

x records written into the list file

x pages written into the list file

When using the /L or /L=filename in conjunction with the Report Writer feature, the message tells you how many pages were written.

If there is no list file defined, you'll receive the message:

Unable to open list file

Chapter 6 contains reference material on all Query commands. The /L and /L=filename switches are discussed there in conjunction with the commands that use them.

Uppercase and Lowercase

If you have an uppercase-only keyboard, you won't have to worry about character case when typing. If you have an upper- and lowercase keyboard, however, you should take note of the following conditions:

1. You may type commands, switches, macros, and filenames in either upper- or lowercase. The Query will accept either form.

   For example, you can type the PATH command in either upper- or lowercase, and get the same response from the Query:

   >> path↵
   Path is PO

   or

   >> PATH↵
   Path is PO

2. You must specify paths and keys according to the conventions of your Query system and your INFOS file. With these entries, the Query *will* discriminate between upper- and lowercase.

   For example, if you were in the PO index and wanted to change to the PARTS index, you must type PARTS in uppercase. The Query won't find it otherwise.

   >> path↵
   Path is PO
   >> path parts↵
   Path is PO

   Since the index name is all uppercase, you must specify it in uppercase.

   >> path PARTS↵
   Path is PARTS

Interrupting Activity on the Screen

Much of your work with the Query involves reading and displaying records on the terminal screen. There may be times when you do not need any more information, or when you realize that you've made a mistake. You can terminate the execution of your terminal's display without terminating your Query session by typing CTRL-C CTRL-A.

↑C↑A

and the Query will respond

CONSOLE INTERRUPT

and return a Query prompt

>>

You can use this control sequence to interrupt any READ, SELECT, NEXT, UP, DOWN, or PRIOR command. The Query will halt all activity, and then you may continue your session.

Correcting Errors

If you make a typing mistake, use the delete (DEL) key to erase the last character you entered. You can erase an entire line of characters by repeatedly deleting single characters, or by using the CTRL-U command. CTRL-U will erase all the characters on the line, except the Query prompt.

Query Commands

The INFOS system Query processor employs a set of fifteen commands that enables you to perform the various Query tasks. You use the commands to communicate interactively with the system; you tell the Query what to do, and the Query will respond with the results.

Query Movement Commands

You use the Query movement commands to travel through the index structure of your INFOS file and find the record(s) you want to select and display. When you initially open your file, you are positioned above the top level index.

The Query's movement commands allow you to move horizontally and vertically in your INFOS file. There are six movement commands: PATH, KEY, UP, DOWN, PRIOR, NEXT. Each one serves a separate and important function for your queries:

PATH	sets or displays where you are currently located within the index structure.
KEY	changes the current position by performing a search based on specific keys within the current index.
UP	moves vertically up through the index structure.
DOWN	moves vertically down through the index structure.
PRIOR	performs horizontal positioning in the current index, moving the position (or path) one key backwards.
NEXT	performs horizontal positioning in the current index, moving the position (or path) one key forwards.

The Query's vertical movement commands, UP and DOWN, in combination with the horizontal movement commands, PRIOR and NEXT, provide a simple, fast, and efficient way of moving through your index structure and establishing the "route" you want. The Query movement commands serve as necessary preconditions to the other Query commands that you use to select and display your records.

Selection and Display Commands

You use the Query's selection and display commands to access records. The Display commands include QFORMAT, RFORMAT, and READ.

QFORMAT	chooses the appropriate qformat for selecting records.
RFORMAT	chooses the appropriate rformat for producing reports.
READ	reads the database record.

As we said earlier, at the heart of the Query is a selection system that enables you to choose records according to pre-established criteria. To do this, you use two commands:

CONDITION	specifies the exact *conditions* for record selection.
SELECT	searches and retrieves specific records according to the set condition.

Other Commands

The index-level separating character is the LEVELMARK. You may find it convenient to change this character while working with the Query. By default, the Query separates the different key names with a colon. If you wish to change the LEVELMARK from a colon to another character, you use the LEVELMARK command.

Use the HELP command whenever you need immediate and brief information about other Query commands. When you type

```
>> HELP↵
```

the Query displays information on your terminal screen. Explanations are available on all the other Query commands.

Each time you complete a Query session, you'll use the BYE command to terminate the session and return to the CLI.

If at any time in the middle of a session you wish to enter the CLI without ending the Query session, type

```
>> CLI↵
```

All these commands and their appropriate switches are explained in detail in the next chapter.

Summary

The Query and the Report Writer feature add an important dimension to the record selection process. Thus far we've introduced the basic concepts involved with both, and we've introduced the different categories of commands. The next chapter will show you how to use the commands and further explain the versatility of the Query.

End of Chapter

Chapter 2
Learning to Use the Query/Report Writer

To explain the actual operation of the INFOS Query facility, we're going to use a retail distributor as the model business application.

The company we've selected represents a "typical" Query application, but the INFOS Query System is not limited to retail datafile operations. You can use the Query facility on any INFOS system database.

The "Typical" Application

Caucus Car Parts is a major distributor of American and imported car parts and accessories. Caucus maintains a huge inventory of over 50,000 items in both a retail store and a nearby warehouse facility. They maintain an INFOS database and use the Query and Report Writer facilities for inventory, billing, and other retail related operations.

The Caucus Car Parts database contains records for all the company's parts, American and imported, as well as the records for the company's current purchase orders to manufacturers. The index structure has three levels. The zero-level index contains the selectors for PARTS, PARTS BY TYPE, and PO (purchase orders).

The level-one index contains the individual PARTS BY TYPE selectors arranged alphabetically, the PARTS keys arranged numerically, and the PO keys arranged numerically. The level-two index contains the PARTS keys referenced by the PARTS BY TYPE subindex. Figure 2-1 shows Caucus's database index structure.

The purpose of the PARTS BY TYPE selectors was to enable the company's personnel to access auto part records quickly and conveniently. The design of the index structure, therefore, served two important purposes: it allowed a convenient means of record retrieval, and it fulfilled a wide range of data processing needs -- inventory, sales, purchase orders, and pricing management.

The Individual Records

The two types of database records, PO records (Purchase Orders) and PARTS records, each contain twelve fields.

The twelve fields in each record type contain enough information so they can be used in a variety of ways. Unlike Caucus's former paper-record system, the database records let different users access information from one source rather than many. For example, prior to the INFOS system, when a customer was interested in a certain car part not readily found on the shelf, the only source of information available to the counter help was the PARTS record. Management didn't really want the customer to see all the information included in the record; the supplier's address and the wholesale price weren't necessary at the point of purchase. Therefore, they designed a qformat called RETAIL which contained most of the PARTS record information, but masked out any information not directly related to a sale. Only information that the counter help might need to make the sale was included in the qformat.

Qformats

This brings us to the specific qformats maintained at Caucus. In addition to the OPEN PO'S qformat, Caucus also consistently maintains a qformat called PARTS which enables any Query user to read the full form of the Parts record type. In other words, one qformat at Caucus contains all the information stored in the Purchase Order record type, and another contains all the information stored in the Parts record types. Caucus also designed other, shorter qformats for specific uses. For instance, one qformat, called RETAIL, contains the information the counter help needed at point of purchase.

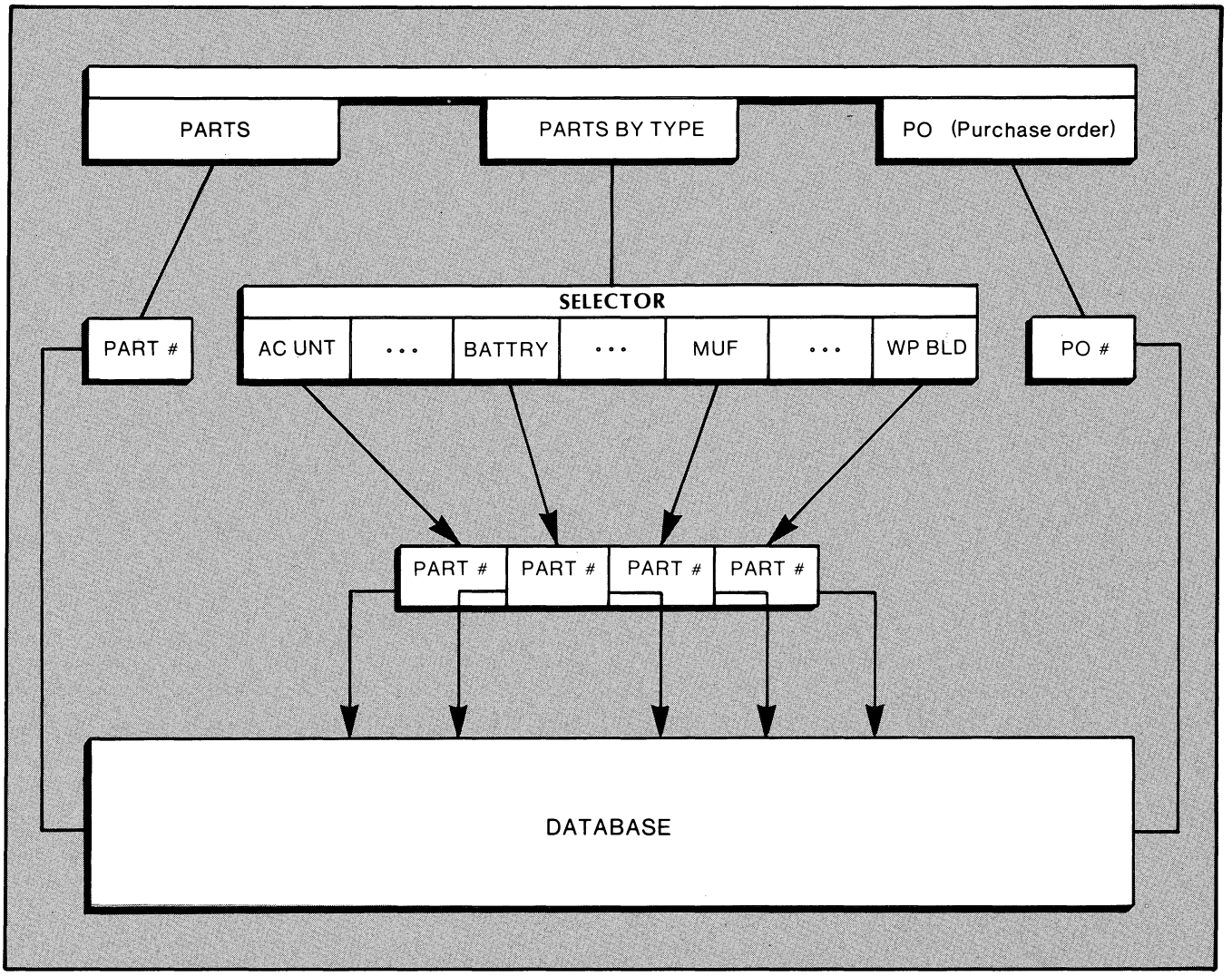


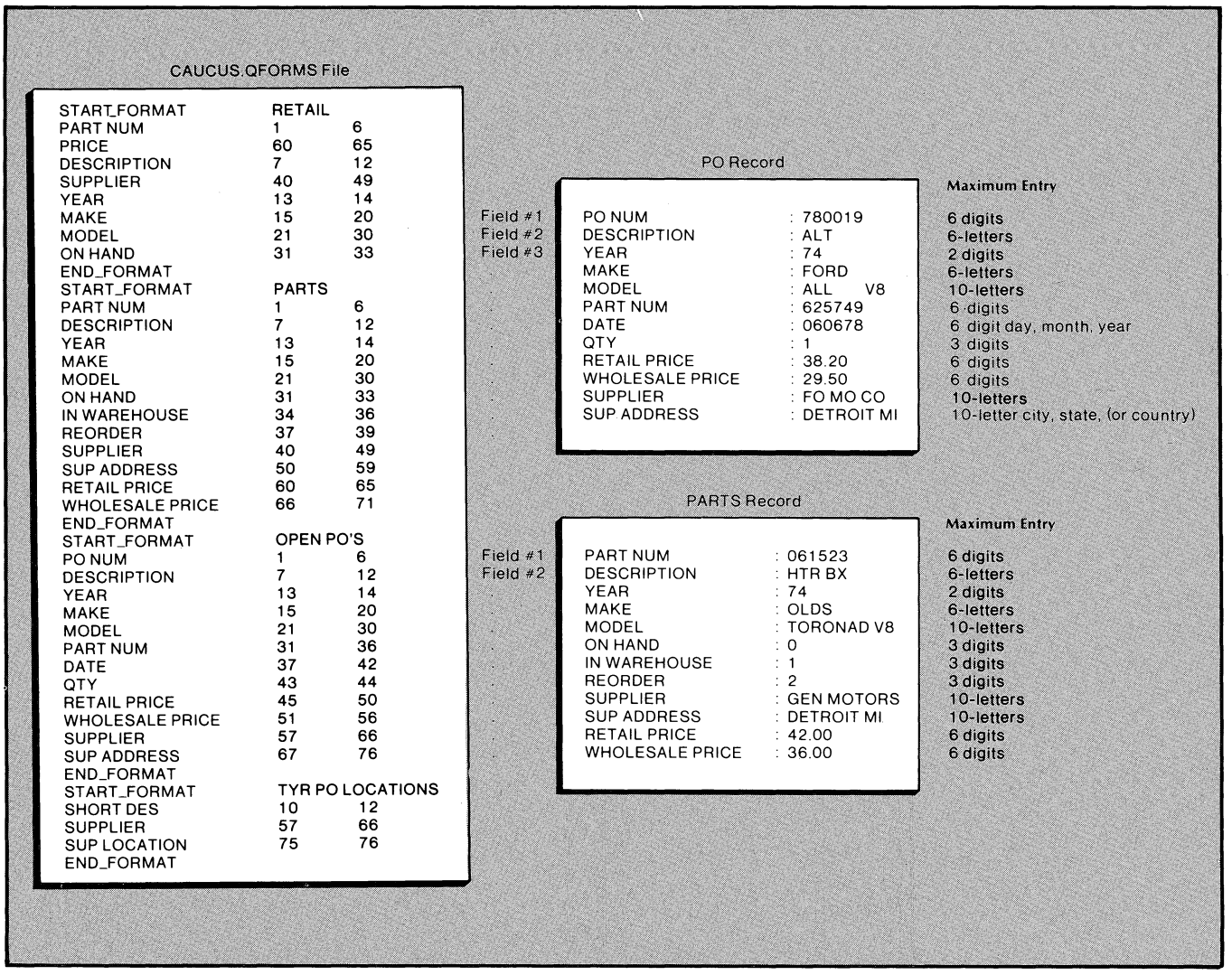
Figure 2-1. Index Structure for Caucus Car Parts

Any Query user with the proper authorization can create, edit, or delete any qformat within the qformat file. The qformats are stored in a separate file called (index).QFORMS, where (index) is the database name. Figure 2-2 shows the contents of the CAUCUS.QFORMS file along with the contents of two sample records.

In the course of using the Query, Caucus's employees frequently create new qformats to display only the specific fields they find neccessary. For example, the person working at the counter often needs to determine what parts are on hand. By using the PARTS qformat, a lot of unnecessary information would be supplied. The counter person doesn't need to know the supplier's address and, in most cases, the quantity in the warehouse. Therefore, by creating the RETAIL qformat that includes just the DESCRIPTION, YEAR, MAKE, MODEL, and ON HAND fields, only needed information is displayed. We'll describe this process in detail in Chapter 3.

Query Commands

As mentioned in Chapter 1, there are fifteen commands that you use to perform all the various Query tasks. In the following sections we'll discuss each command and demonstrate its function, as used by the Caucus database. As mentioned in the Preface, you have a copy of this database on tape and can try out the examples for yourself.



SD-01337

Figure 2-2. Contents of Caucus.QFORMS File and Two Sample Records

First we'll talk about the movement commands, then the selection and display commands. And, finally, we'll discuss the other miscellaneous commands you'll use with the Query. Chapter 6 contains reference material on all the commands, switches, and arguments.

Movement Commands

As explained in Chapter 1, at the core of the INFOS system is the index structure. You use the Query movement commands to travel through this index structure. The movement commands let you travel horizontally and vertically in your INFOS file.

When you want to move to a different position within the index structure, you change your path. When you want to move to a different position within a current subindex, you change your key. You can change the key through a key search or an approximate key search, as well as by moving forwards or backwards within the subindex.

A key search tells the Query you are looking for a specific key or a key that begins with the specified letters (INFOS calls this a generic key search). You indicate the key and ask the Query to locate an entry that matches it.

An approximate search is one in which you ask the Query to find either an exact match to the key you specify, or if there is no exact match, the next following key.

In the following sections we discuss each movement command in detail. Chapter 6 contains reference material on all Query commands.

PATH command

The PATH command sets or displays the current location within the index structure. Use it to move to different parts of the database or to find out where you are.

When you first log on to Query and type PATH, the system will respond with

Path is above the index

If you want to move to a certain index level, you use the name of the index as an argument:

```
>> PATH PARTS↵
Path is PARTS
```

You may also use an argument with the PATH command when you want to move to a desired key position anywhere within the index structure. In this case, the argument is a series of key fragments (abbreviations of a key) separated by a colon (:). The Query then performs a generic keyed search using the multilevel key you specified. If that multilevel search fails -- in other words, if you specify a key fragment that doesn't belong to the key sequence -- the Query will position the path at the last key it successfully found.

For example:

```
>> PATH PARTS↵
Path is PARTS
>> PATH PARTS BY↵
Path is PARTS BY TYPE
>> PATH PARTS BY TYPE:CARB↵
Path is PARTS BY TYPE:CARB
>> PATH PARTS BY TYPE:HSE↵
Path is PARTS BY TYPE
```

The key fragment HSE doesn't belong to any key sequence, so the Query positioned the path to the last key fragment it successfully found, PARTS BY TYPE.

Figure 2-3 shows how the PATH command moves you through the index.

You can specify only the number of key fragments that the path would contain to its lowest level (the person who creates the INFOS file decides the number of levels). If you specify additional paths, the system will respond with

Path has too many levels

Your previous path will not change.

For example:

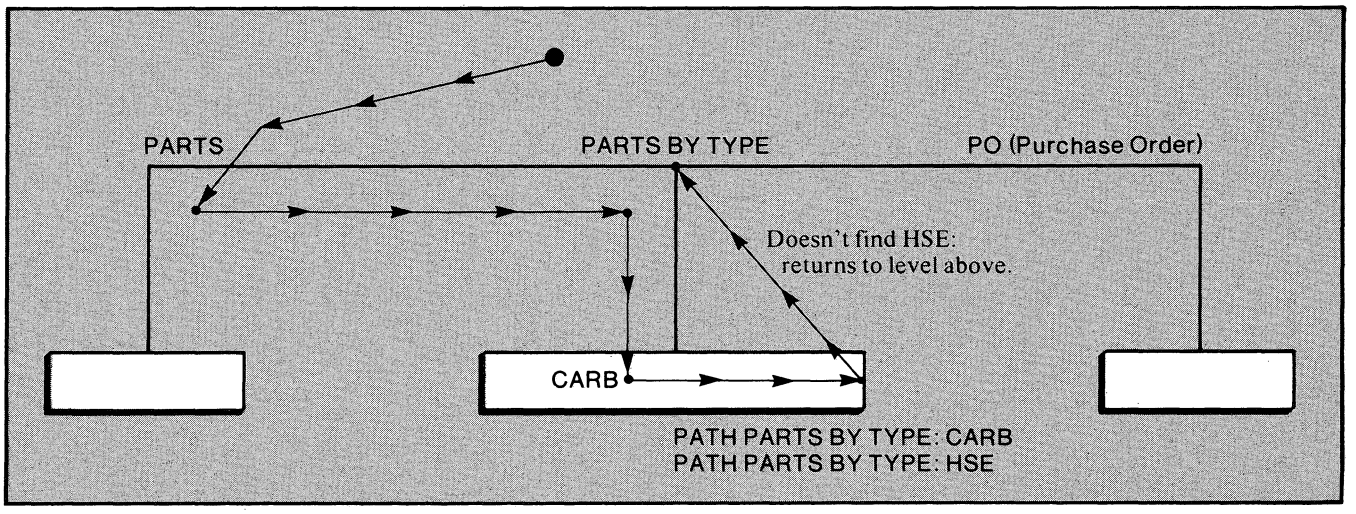
```
>> PATH↵
Path is PARTS
>> PATH PARTS BY TYPE:AC UNT:599603:222↵
Path has too many levels
```

KEY Command

You use the KEY command to change the current path by performing a keyed search in the current index. You can also specify a key fragment as an argument to the KEY command if you want to do a generic search of the index. In either case, if the search succeeds, the system will change the current path. If, however, you specify an unknown key or an incorrect key fragment, the Query will respond with

Keyed search was unsuccessful

and the path will remain unchanged.



SD-01338A

Figure 2-3. The PATH Command

For example:

```
>> PATH↓
Path is PARTS BY TYPE:OIL FT
>> KEY CARB↓
Path is PARTS BY TYPE:CARB
>> KEY XXX↓
Keyed search was unsucessful
```

In many cases, you'll find that you don't know whether or not a specific key or a key fragment exists. You might know the general location of the key, but not its exact sequence. In these cases, you can use the command switch /APPROX to perform an approximate search.

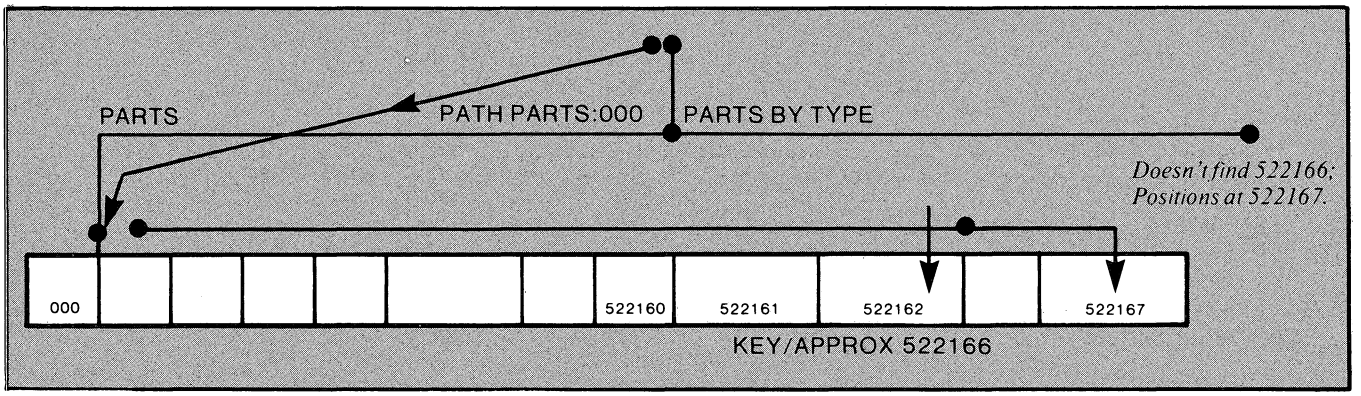
For example, if you're in the PARTS index and you're not sure if there's a key for 522166, you can use the KEY/APPROX command to find either that key or the next following one. The Query then repositions the current path to the key it locates.

```
>> PATH↓
Path is PARTS:056793
>> KEY/APPROX 522166↓
Path is PARTS:528831
```

Figure 2-4 shows how the KEY/APPROX command works.

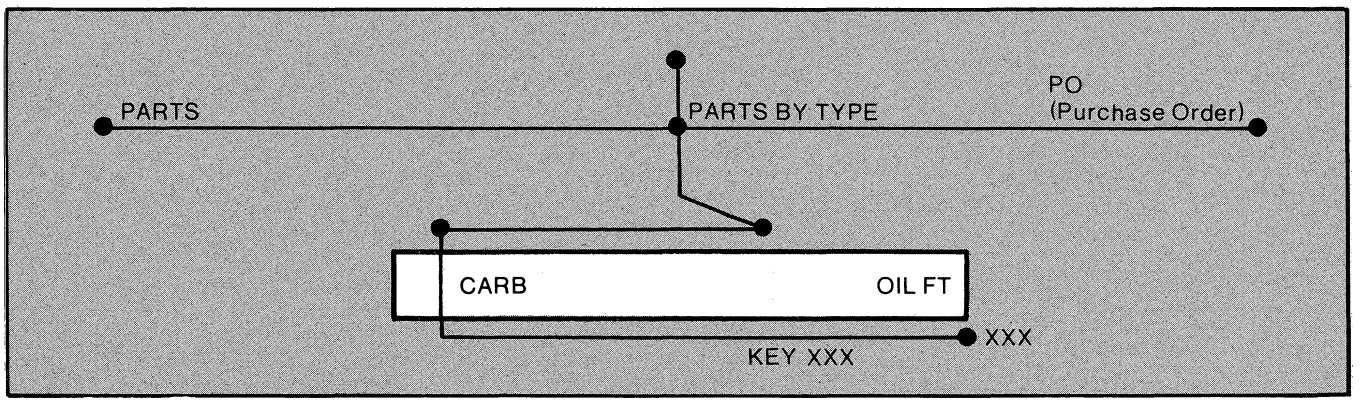
Don't confuse the KEY command with the PATH command. While the PATH command searches through the entire index structure, the KEY command searches only the current subindex. Oftentimes, during a Query session, you might alternate between the KEY and PATH commands. That is, you may PATH to a certain index level, and then KEY to a desired position in that index level. Keep in mind that the PATH command changes your position within the entire index structure, while the KEY command only changes your position within the current index level.

Figure 2-5 illustrates how the PATH and KEY commands work.



SD-01338B

Figure 2-4. The KEY/APPROX Command



SD-01338C

Figure 2-5. The PATH and KEY Commands

UP and DOWN Commands

Use the UP and DOWN commands whenever you want to move vertically (from subindex level to subindex level) in your file. The DOWN command will set the current path by descending one subindex level; it will also retrieve the first key in the subindex. The Query will then append the new key to the path and display the current path. You can use the /SUBINDEXES switch with the DOWN command to determine if there is a subindex beneath the retrieved key.

For example, when you first log on to Query:

```
>> PATH↓
Path is above the index
>> DOWN/SUBINDEXES↓
Path is PARTS
*** Subindex beneath this entry
```

The DOWN command always positions the path at the *first* key in an index. If you are in the middle of an index and you move down to the next subindex, you'll be positioned on the first key in that subindex.

For example:

```
>> PATH↓
Path is PARTS BY TYPE
>> DOWN/SUBINDEXES↓
Path is PARTS BY TYPE:AC UNT
*** Subindex beneath this entry
```

The UP command moves the current path to the next higher subindex in that path. In effect, the UP command removes the last current key in the path. You can use numeric arguments with the UP command to move the specified number of index positions. Since you will be moving from a subindex within an index, you will not need to use the /SUBINDEXES switch.

Note, however, that if you specify a numeric argument to the UP command that exceeds the number of higher index levels, the Query will move to a position above the highest index level and print the message

Path is above the index

If you then type another UP command, you'll receive the message

Sorry, you are already at the top

While working with the Query, you may find yourself positioned in the middle of a long index. Using an UP-DOWN command combination is a quick way to move back to the beginning of the index. By using these two commands in sequence, you'll first move to the beginning of the next higher index level and then descend to the beginning of the index level from which you started.

For example:

```
>> PATH↓
Path is PARTS BY TYPE:OIL FT
>> UP↓
Path is PARTS BY TYPE
>> DOWN↓
Path is PARTS BY TYPE:AC UNT
```

The Query's vertical movement commands, UP and DOWN, in combination with the horizontal movement commands, PRIOR and NEXT, provide a simple, fast, and efficient way of moving through your index structure and establishing the record path you want.

NEXT and PRIOR Commands

The NEXT and PRIOR commands change the current path by performing horizontal positioning in the current index. Without an argument, the NEXT command moves the path forward one key position, retrieves the key, and then prints the complete path. The PRIOR command performs a similar function, except it moves the path back one key.

For example:

```
>> PATH↓
Path is PARTS BY TYPE:DIS CP
>> NEXT↓
Path is PARTS BY TYPE:EN CRK
>> PRIOR↓
Path is PARTS BY TYPE:DIS CP
```

If you include a number as an argument, the NEXT and PRIOR commands will move forward and backward more than one position. The Query will read the number, perform the movement the specified number of times, and echo the keys on the terminal as it moves through the index.

For example:

```
>> PATH↓
Path is PARTS BY TYPE:DIS CP
>> NEXT 3↓
        EN CRK
        FN BLT
Path is PARTS BY TYPE:FP
>> PRIOR↓
Path is PARTS BY TYPE:FN BLT
```

You may also use the PRIOR and NEXT commands to list the entire contents of a given index. First specify a PATH to that index, followed by NEXT 32767 (32767 is the maximum number Query accepts as an argument).

For example, in Caucus's index, you can completely list all the keys in the level-one index by typing

```
>> PATH PARTS BY TYPE↓
Path is PARTS BY TYPE
>> DOWN↓
Path is PARTS BY TYPE:AC UNT
>> NEXT 32767↓
        AIR FT
        ALT
        BATTRY
        BRA DR
        BRA LN
        BRA LT
        BRA PD
        CARB
        CM
        CNV MT
        CNV TP
        CNV W
        DIS CP
        EN CRK
        FN BLT
        FP
        GS CP
        GS TNK
        GS TRT
        HD GSK
        HD LGT
        HORN
        HTR BX
        HTR HS
        IG PTS
        MUF
        OIL
        OIL FT
        OILTRT
        RD TYR
        SE BLT
        ST WHL
        TL PIP
        TM CHN
        TM LGI
        TRANS
        WHL CY
        WP BLD
Alas - no more entries in this subindex
```

The Query responds by listing out all the PARTS BY TYPE selectors (that follow AC UNT), ranging from AIR FT (air filter) to WP BLD (wiper blade). When the Query reaches the end of an index, it will type out the message

Alas - no more entries in this subindex

The Query also offers a convenient feature to check whether you have subindexes beneath the key you want to retrieve. By using the /SUBINDEXES switch with either the NEXT or PRIOR command, the Query will identify the presence of existing subindexes with the message

**** Subindex beneath this entry*

Using the /SUBINDEXES switch is another way to determine the size and the scope of your index structure. You can use it to list the contents of an index; if you later want to descend to the next index level, you'll know which key to reference.

For example:

```
>> PATH↓
Path is PARTS BY TYPE:GS CP
>> NEXT/SUBINDEXES↓
Path is PARTS BY TYPE:GS TNK
*** Subindex beneath this entry
```

Selection and Display Commands

Once you have created a path to the record(s), you can use the Query Selection and Display commands to access those records. In the following sections we describe how to use these commands once qformats and rformats have been established. In Chapter 3, we describe how to set up your own qformats and rformats, and how to tailor the Query facility to fit your individual application.

QFORMAT Command

You use the QFORMAT command with your existing Query system to choose the appropriate qformat for displaying records in your file. A qformat is simply a list of the relevant fields you want to access. You'll use different qformats for outputting records and for formulating selection criteria. For example, as we mentioned earlier, the Query facility at Caucus Car Parts contains two separate record types -- PARTS and POS.

(Users of Query Rev. 1 should note that this Rev. of the Query uses the QFORMAT command in addition to the FORMAT command; both commands perform the same function.)

The Caucus database contains four qformats: OPEN PO'S, PARTS, RETAIL, and TYR PO LOCATIONS. Before you can set a CONDITION and SELECT records, you must specify which qformat you want to use. You do not have to select a qformat if you are only READing records. However, without an activated qformat, the Query will display the record across the screen as it appears in the database in one straight line.

The QFORMAT command without arguments displays the name of the currently active qformat, if any.

When you enter the Query and type

```
>> QFORMAT↵
```

the system will respond with

There is no current qformat selected

If you know the qformat that you want to use, simply type

```
QFORMAT <name>↵
```

where <name> is a previously defined qformat.

For example:

```
>> QFORMAT OPEN PO'S↵
```
The current qformat is OPEN PO'S

The QFORMAT < name > command activates the qformat you specify. If you don't know which qformat to choose or what qformats your Query file contains, type

```
>> QFORMAT/NAMES↵
```

The /NAMES switch tells the Query to display the names of all the currently defined qformats. The qformat names will appear on your terminal in the order that they appear in the .QFORMS file. For example:

```
>> QFORMAT/NAMES↵
```
The following qformats have been defined on this file:
RETAIL
PARTS
OPEN PO'S
TYR PO LOCATIONS

Once you've activated an available qformat, you can use the /DISPLAY switch to examine the contents of the individual qformat:

```
>> QFORMAT PARTS↵
```
The current qformat is PARTS
```
>> QFORMAT/DISPLAY↵
```

PART NUM	*1*	*6*	*ASCII*
DESCRIPTION	*7*	*12*	*ASCII*
YEAR	*13*	*14*	*ASCII*
MAKE	*15*	*20*	*ASCII*
MODEL	*21*	*30*	*ASCII*
ON HAND	*31*	*33*	*ASCII*
IN WAREHOUSE	*34*	*36*	*ASCII*
REORDER	*37*	*39*	*ASCII*
SUPPLIER	*40*	*49*	*ASCII*
SUPP ADDRESS	*50*	*59*	*ASCII*
RETAIL PRICE	*60*	*65*	*ASCII*
WHOLESALE PRICE	*66*	*71*	*ASCII*

In this example, the QFORMAT/DISPLAY command lists all the fields (including description, start byte, end byte, and display mode) contained in the PARTS qformat.

If, at any time, you want to deactivate the current qformat without changing to a different one, use the /CLEAR switch:

```
>> QFORMAT/CLEAR↵
```
There is no current qformat selected

The /CLEAR switch deactivates the current qformat and returns to the default display mode of one record per line.

We describe how you can create, edit, and delete qformats in Chapter 3; but for now, just realize that you must use the QFORMAT command and its switches to

1. activate a qformat,
2. change the current qformat,
3. display the contents of a qformat, or
4. list the names of the defined qformats.

RFORMAT Command

You use the RFORMAT command to choose the appropriate report format that you want the Report Writer to use. A report format, or rformat, is simply a description of the way you want data organized in a report produced by the Report Writer.

Different rformats produce different reports, as you'll see in Chapter 3 when we talk about creating rformats.

Once you've activated an rformat with the RFORMAT command, you can use the /RFORMAT switch to display a report on the terminal screen or produce a hard copy report on the line printer.

As with the QFORMAT command, by typing RFORMAT without arguments, the name of the current rformat appears on the screen. For example:

```
>> RFORMAT↵
```
There is no current rformat selected

If you know the name of the rformat you want to use, simply type

```
>> RFORMAT <name>↵
```

where <name> is a previously defined rformat.

One rformat at Caucus Car Parts is STOCK VALUE. Therefore, if you type

```
>> RFORMAT STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
```

RFORMAT < name > activates the rformat you specify. If you don't know which rformats exist or which one to choose, use the /NAMES switch.

For example:

```
>> RFORMAT/NAMES↵
The following rformats have been defined:
        STOCK VALUE
        OLDIES
```

The RFORMAT/DISPLAY command lists all the Format Descriptor Lines contained in the active rformat. At Caucus, if you use the /DISPLAY switch, you'll see on your terminal what you see in Figure 2-6.

```
START_REPORT    STOCK VALUE
HEADER  1       C       "As the Kid Goes for Broke"
!
!       This produces a report with the total value of the current
!       warehouse stock
!
QFORMAT         PARTS
LIN/PG          60
COL/LIN         80
DEFINE  DOLLAR VALUE    IN WAREHOUSE * WHOLESALE PRICE
PICTURE DOLLAR VALUE    $$$$$99V.99
HEADER  2       C       "CAUCUS CAR PARTS"
HEADER  2       70      PAGE
HEADER  4       C       "Dollar Value of Warehouse Inventory"
HEADER  4       70      DATE
HEADER  6       5       "Description"
HEADER  7       5       "**********"
DETAIL  1       5       DESCRIPTION
BREAK   DESCRIPTION     NO_REP
BREAK   DESCRIPTION     POST_BREAK_SPACE
BREAK   DESCRIPTION     40      "Value of"
BREAK   DESCRIPTION     49      DESCRIPTION
BREAK   DESCRIPTION     56      "Stock ="
BREAK   DESCRIPTION     66      TOTAL ( DOLLAR VALUE )
HEADER  6       20      "Part Number"
HEADER  7       20      "**********"
DETAIL  1       20      PART NUM
HEADER  6       35      "Stock"
HEADER  7       35      "*****"
DETAIL  1       35      IN WAREHOUSE
HEADER  6       45      "Cost"
HEADER  7       45      "******"
DETAIL  1       45      WHOLESALE PRICE
HEADER  6       60      "Dollar Value"
HEADER  7       60      "***********"
DETAIL  1       60      DOLLAR VALUE
TOTAL   3       30      "Total Value of Warehouse Stock"
TOTAL   3       65      TOTAL ( DOLLAR VALUE )
END_REPORT
```

Figure 2-6. Using the /DISPLAY Switch

If, at any time you want to deactivate the current rformat without changing to a different rformat, use the /CLEAR switch:

```
>> RFORMAT/CLEAR↓
There is no current rformat selected
```

We describe how you can create, edit, and delete rformats in Chapter 3; but for now, just realize that you must use the RFORMAT command and its switches to

1. activate a rformat,
2. change the current rformat,
3. display the contents of a rformat, or
4. list the names of the defined rformats.

READ Command

You read a database record using the READ command. If no qformat is active, the Query reads and displays the data as it appears on the record in one straight line. If you have activated a qformat, the READ command reads and displays the record according to the specified qformat. If you want the record displayed according to a specified rformat, you must use the /RFORMAT switch, as explained later. You should remember, though, that the READ command will only read a record after you have established a path to a key linked to a database record. If your path only extends to a key that doesn't have a database record, the Query will respond

```
>> READ↓
Key has no database record
```

Therefore, if you receive this message, you must use the Query movement commands to reposition the path to a valid key.

You can use numeric arguments with the READ command. The Query then READs the number of records you specify, starting from the current path.

Your index structure affects the results of a numeric argument to a READ command. For instance, the index PARTS BY TYPE has a selector index above the actual data. In this case, when you travel down the index structure to the first selector, AC UNT, and ask the Query to READ 3 records, it will read all the records associated with *that* selector. To read all the records from this subindex or more than just the records associated with one selector, you must use a /TRAVERSE switch, as shown in Figure 2-7. You don't need to use this switch in the PARTS index because there is not a selector index above the actual data.

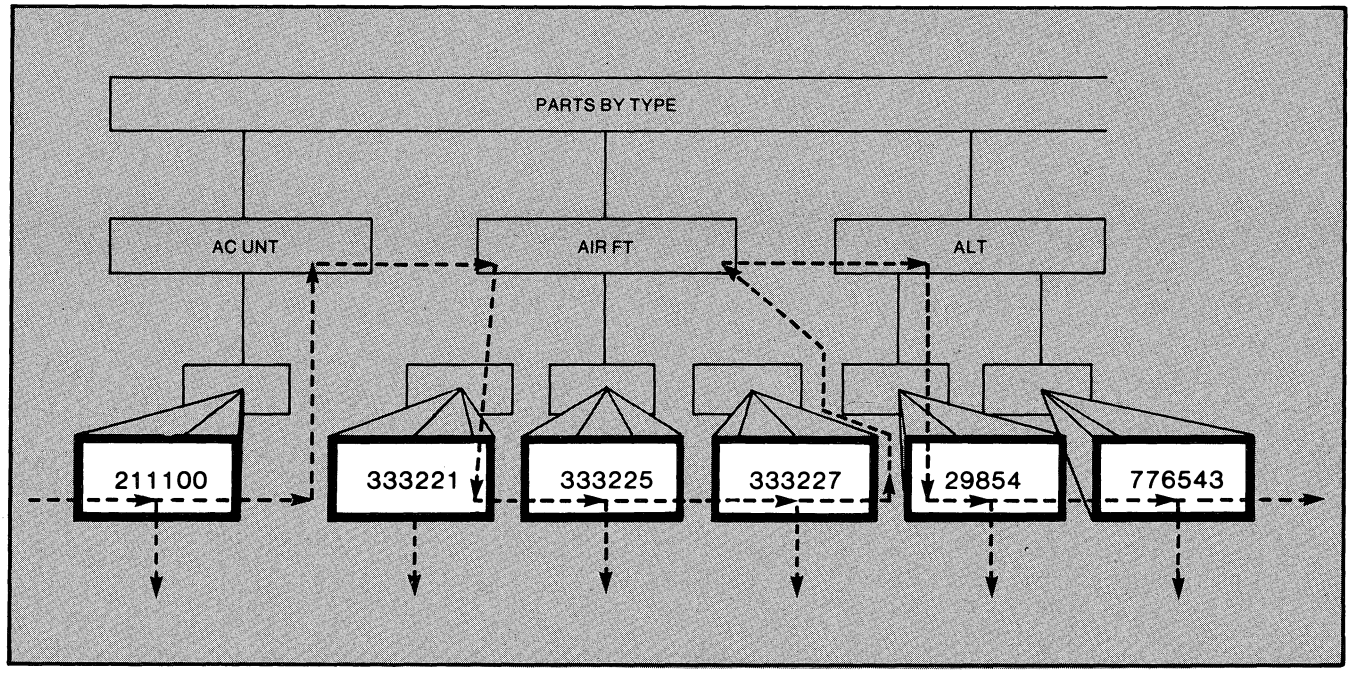


Figure 2-7. The READ/TRAVERSE Command

For example, if your path is PARTS BY TYPE:AC UNT, and you move DOWN and ask the Query to READ 3 records, you'll only get one record:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
>> READ 3↵

PART NUM     : 211100
PRICE        : 89.99
DESCRIPTION  : AC UNT
SUPPLIER     : POR AUDI
YEAR         : 75
MAKE         : AUDI
MODEL        : FOX
ON HAND      : 2
>>
```

However, if you follow the same steps, using the /TRAVERSE switch, the Query will return three records:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
>> READ/TRAVERSE 3↵

PART NUM     : 211100
PRICE        : 89.99
DESCRIPTION  : AC UNT
SUPPLIER     : POR AUDI
YEAR         : 75
MAKE         : AUDI
MODEL        : FOX
ON HAND      : 2

PART NUM     : 333221
PRICE        : 1.95
DESCRIPTION  : AIR FT
SUPPLIER     : GEN MOTORS
YEAR         : 72
MAKE         : CHEV
MODEL        : IMPALA 225
ON HAND      : 12

PART NUM     : 333225
PRICE        : 1.89
DESCRIPTION  : AIR FT
SUPPLIER     : GEN MOTORS
YEAR         : 73
MAKE         : CHEV
MODEL        : BELAIR 225
ON HAND      : 17
```

If you want to READ a series of records starting with your path up to a known record, use the /UPTO switch. This switch tells the Query to read all the records until it encounters the specified approximate key. If you use either READ option, READ n or READ/UPTO key, the Query will change the current path as it reads succeeding records. If you want to keep the current path stationary, use the /RETURN switch to return to the starting position once you have read the specified records. After all your records are displayed, the /RETURN switch also directs Query to print the returned path.

For example:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> PATH PARTS↵
Path is PARTS
>> DOWN↵
Path is PARTS:056793
>> READ/RETURN/UPTO 06↵

PART NUM     : 056793
PRICE        : 48.79
DESCRIPTION  : CM
SUPPLIER     : FO MO CO
YEAR         : 72
MAKE         : FORD
MODEL        : PINTO V6
ON HAND      : 0

PART NUM     : 057211
PRICE        : 16.23
DESCRIPTION  : BRA PD
SUPPLIER     : VOLVO AM
YEAR         : 69
MAKE         : VOLVO
MODEL        : CNV 18005
ON HAND      : 4

PART NUM     : 057217
PRICE        : 16.78
DESCRIPTION  : BRA PD
SUPPLIER     : VOLVO AM
YEAR         : 70
MAKE         : VOLVO
MODEL        : CNV 18005
ON HAND      : 2
```

PART NUM : 057219
PRICE : 16.57
DESCRIPTION: BRA PD
SUPPLIER : VOLVO AM
YEAR : 72
MAKE : VOLVO
MODEL : CNV 18005
ON HAND : 4

PART NUM : 061523
PRICE : 42.00
DESCRIPTION: HTR BX
SUPPLIER : GEN MOTORS
YEAR : 74
MAKE : OLDS
MODEL : TORONAD V8
ON HAND : 0
Path is PARTS:056793

The READ command simply displays the contents of a record on the terminal. To store those records in a file, use either the /L or the /L=filename switch, as explained in Chapter 1.

If you use the /RFORMAT switch, the Query uses the currently selected rformat to produce its output. In other words, the records read will be displayed in the form of a report. For example, if you wanted to see seven specific records in a report, you would first set an rformat, then read the records using the /RFORMAT switch:

```
>> PATH
Path is PARTS:056793
>> RFORMAT STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
>> READ/RFORMAT 7↵
```

Figure 2-8 shows what would then be displayed on your terminal.

As the Kid Goes for Broke
CAUCUS CAR PARTS

Page 1

Dollar Value of Warehouse Inventory

11/13/79

Description	*Part Number*	*Stock*	*Cost*	*Dollar Value*	
CM	*056793*	*0*	*39.50*	*$0.00*	
			Value of CM	*Stock =*	*$ 0.00*
BRA PD	*057211*	*3*	*13.12*	*$39.36*	
	057217	*5*	*13.26*	*$66.30*	
	057219	*7*	*13.21*	*$92.47*	
			Value of BRA PD Stock =		*$198.13*
HTR BX	*061523*	*1*	*36.00*	*$36.00*	
			Value of HTR BX Stock =		*$36.00*
BRA LN	*062251*	*4*	*4.11*	*$16.44*	
			Value of BRA LN Stock =		*$16.44*
WHL CY	*066234*	*2*	*10.11*	*$20.22*	
			Value of WHL CY Stock =		*$20.22*
			Total Value of Warehouse Stock		*$270.79*

Figure 2-8. Using the /RFORMAT Switch

You can also use a combination of switches to append the same report to the current list file or a named file. For example:

```
>> READ/RFORMAT/L=REPORT_TEST 7↓
1 pages written into the list file
```

Then, when you finish your Query session, type the file REPORT_TEST to see the same report that appeared at your terminal.

If you have an rformat that defines a report with up to 80 columns per line (see Chapter 5), you can use the /SCREEN switch. (Note that one character fits in each column on a line.) The /SCREEN switch uses the special features of the DASHER™ Display terminal to display the report screen by screen. The Report Writer will first display the header information. These lines will remain on your terminal screen as long as the report is being displayed. The actual data you want included in the report will fill up the screen until there is no more room. (Your terminal screen will display 22 lines.) When the 22 lines are full, the Report Writer asks if you want to see another screenful or stop. Again, note that the report must not contain more than 80 columns per line. The Query won't allow you to use the /SCREEN switch if you've defined a wider report.

For example, if you wish to read all the records in the Caucus database using the STOCK VALUE rformat, you could set the rformat and ask the Query to READ ALL. The records would zip by on your screen. Or, you could use the /SCREEN switch and see one screenful of records at a time.

You use the READ command and its various switches whenever you need to access information directly from your database. Once you move to a readable key with the PATH command, you can either display your INFOS records at your terminal for immediate access, read them into a file for later use, or both.

CONDITION and SELECT Commands

So far, we've discussed how you can use the Query to move through indexes and read database records. But much of your work with the Query will go beyond positioning and displaying. As we said earlier, at the heart of the Query is a selection system that enables you to choose records according to pre-established criteria. You use the CONDITION command to specify the exact *conditions* for selection, and you use the SELECT command to tell the Query to search and retrieve those specific records from your subindexes.

Using the Query's CONDITION and SELECT commands is like guiding a pair of intelligent "eyes" that scans through your indexes and "sees" only those records that you need.

You must do three things to guide the Query's eyes:

1. Put the appropriate glasses on the eyes so they see only those record parts that you want to view; use the QFORMAT command to grind the lenses.

2. Tell the brain to remember the criteria for selection by using a CONDITION command.

3. Point the eyes towards the index you want to scan, and tell the eyes how many records you you want them to read. You do the pointing with the Query movement commands; and you do the scanning with the Query SELECT command.

Let's say that you want to select (from a subindex) all the records that contain FORD parts. You must first declare a condition: MAKE="FORD", where the first item, MAKE, is a dataname (a title of a record field), the second item, = (equals), is a relational operator, and the third item, FORD, is a constant (a database value). All the Query CONDITIONS you specify will be relations between various datanames and/or constants.

Before the Query's eye can scan through the subindex, you must activate a qformat (grind the glasses) to focus the Query's vision.

In this example, you would select the qformat PARTS because it contains all the fields that you need to access, including the dataname MAKE, from your CONDITION statement. Once you've activated this qformat, you tell the Query's eyes to start searching through the subindex. The Query's eyes will begin inspecting at the current path, selecting only those records that it can see clearly (qformat) and that register with its memory (CONDITION). Figure 2-9 illustrates the operation of the Query's eyes.

CONDITION Command

You use the CONDITION command to set or display selection criteria for which the SELECT command will search. Without an argument, CONDITION displays the last condition you defined.

For example:

```
>> CONDITION↓
Condition is:    YEAR="76"
```

If you have not specified a CONDITION, the system will respond with

No condition is specified

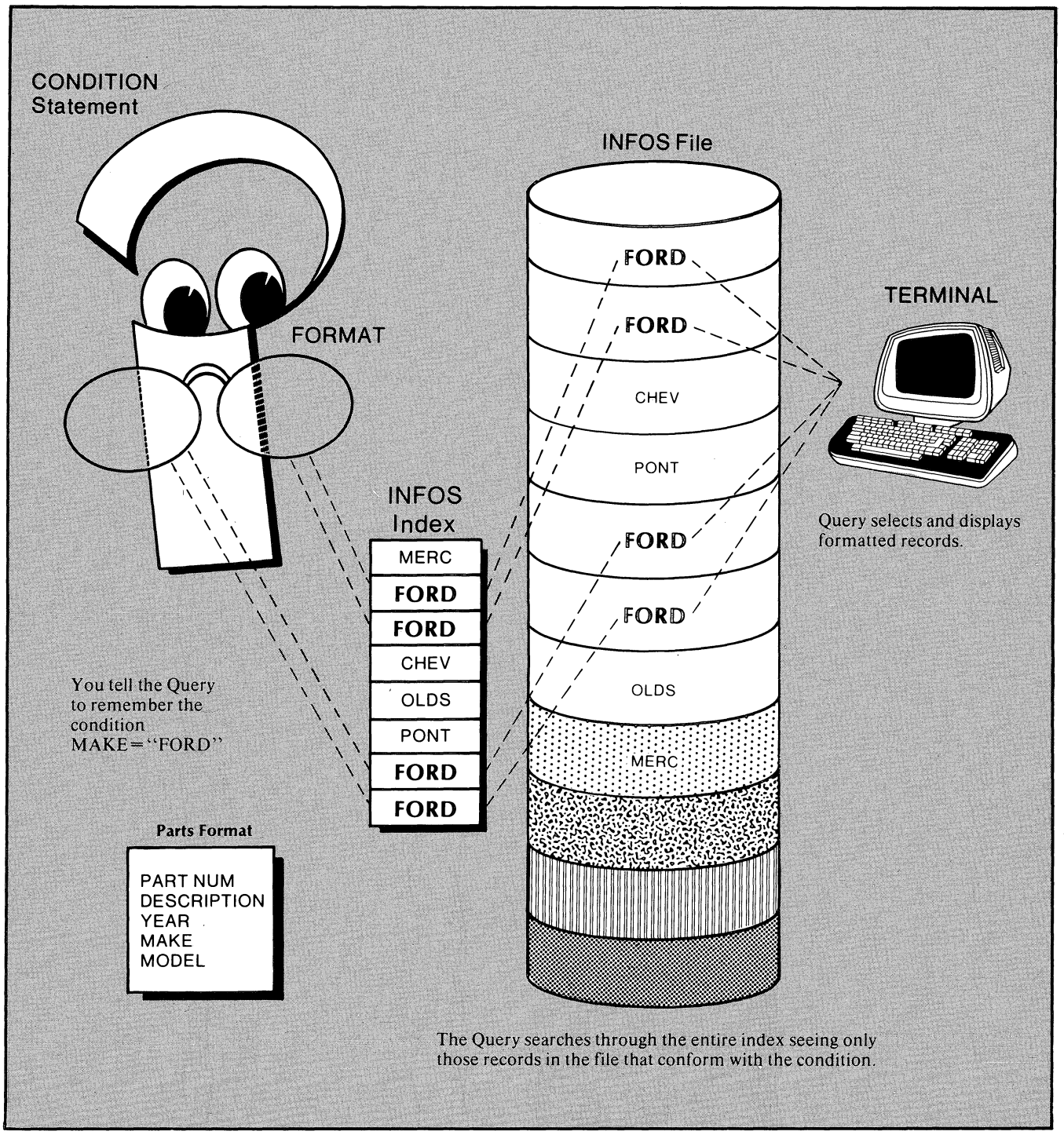


Figure 2-9. The Query's Eyes

An argument to the CONDITION command forms a simple relation between various record parts (fields) and/or constants. You can think of the CONDITION command with a defined relation as a *CONDITION STATEMENT.* After you have established one CONDITION STATEMENT, you can append others to it.

These appended CONDITIONs further narrow the scope of your search. You can append CONDITIONs using either the /AND or the /OR switch. Both switches logically combine one CONDITION with another. For example, if the first CONDITION you specify is MAKE="FORD", you can append a second condition that specifies the year by typing

```
>> CONDITION MAKE="FORD"↵
Condition is: MAKE = "FORD"
>> CONDITION/AND YEAR="78"↵
Condition is: MAKE = "FORD"
        AND   YEAR="78"
```

String constants must be enclosed in quotation marks. If you make a mistake by either defining a CONDITION that doesn't correspond to your qformat or forgetting to enclose the string constant in quotation marks, the Query will send you an error message:

Not a legal variable -- PHRASE: 1 (or 2)

where *PHRASE 1* represents the phrase before the relational operator and *PHRASE 2* represents the phrase after the relational operator. For example:

```
>> QFORMAT PARTS↵
The current qformat is PARTS
>> CONDITION MAKE=FORD↵
Not a legal variable -- PHRASE: 2
```

In this example, the error occurs in *PHRASE 2,* FORD, which is not enclosed in quotation marks. However, if the constant is a number, it does not have to be enclosed in quotation marks.

Also be sure to activate a qformat before specifying a condition. If you forget, the Query will remind you by typing

There is no current qformat selected

You can specify a variety of relationships in your CONDITION statements by using one or more of the following relational operators:

=	equal to
< >	not equal to
>	greater than
<	less than
= >	equal to or greater than
= <	equal to or less than
> =	greater than or equal to
< =	less than or equal to

For example, all the following relationships are legal:

Dataname	**Relational Operator**	**Constant**
MAKE	=	"FORD"
MODEL	< >	"MSTG"
YEAR	>	60
RETAIL PRICE	<	"20.00"
WHOLESALE	> =	12.00
PART NUM	< =	"950861"

Additionally, you can specify CONDITION STATEMENTS between datanames.

For example:

Dataname	**Relational Operator**	**Dataname**
MODEL	< >	MAKE
ON HAND	< =	IN WAREHOUSE
WHOLESALE PRICE	=	RETAIL PRICE
REORDER	> =	ON HAND

Because the Query recognizes the datanames as parts of a record, don't enclose them in quotation marks.

This is only a brief description of the CONDITION command. In Chapter 3, we illustrate further uses (and wider applications) of the CONDITION STATEMENT.

Once you have finished using a CONDITION STATEMENT, you can deactivate it by either changing the current QFORMAT or using the /CLEAR switch. Changing the qformat at any time automatically deactivates the CONDITION STATEMENT. When you use the CONDITION/CLEAR command, the Query responds

No condition is specified

Once you have cleared the old CONDITION, you can declare a new one.

SELECT Command

After you have specified a CONDITION, use the SELECT command to search through the current subindex for conforming records.

The search starts from the current path and continues forward to the end of the subindex or until the Query locates the desired record. Any record selected by the Query is displayed according to the active qformat. Without an argument, the SELECT command searches the subindex until it finds the first matching record. The Query displays it on the terminal and the search is over.

For example:

```
Path is PARTS:056793
>> QFORMAT RETAIL↓
The current qformat is RETAIL
>> CONDITION MAKE = "FORD"↓
Condition is:          MAKE = "FORD"
>> CON/AND YEAR = 76↓
Condition is:          MAKE = "FORD"
AND                    YEAR = 76
>> SELECT↓

PART NUM : 333229
PRICE : 23.40
DESCRIPTION : GS TNK
SUPPLIER    : FO MO CO
YEAR        : 76
MAKE        : FORD
MODEL       : CNV IMP V8
ON HAND     : 2
```

You must specify a CONDITION before you can issue a SELECT command. And, you must activate a qformat before you can specify a CONDITION. Therefore, if you want the Query to ignore the current qformat and display the records as they appear in the database (in one straight line), use the /NOFORMAT switch.

For example:

```
Path is  PARTS:056793
>> QF/CLEAR↓
There is no current qformat selected
>> SELECT↓
No condition is specified
>> COND MAKE = "FORD"↓
There is no current qformat selected
>> QFORMAT RETAIL↓
The current qformat is  RETAIL
>> COND MAKE = "FORD"↓
Condition is:          MAKE = "FORD"
>> CON/AND YEAR = 76↓
Condition is:          MAKE = "FORD"
               AND     YEAR = 76
>> SEL/NOFORMAT↓
333229GS TNK76FORD  CNV IMP V8  2  1  3FO MO CO  DETROIT MI 23.40 18.10
```

You can attach numeric arguments to the SELECT command so the Query will search from the current path until it finds that number of matching records. If the subindex contains no matching records, the system will respond

NO records selected

You may often want the Query to search through the entire subindex and display all the matching records. To conduct an exhaustive search, type

```
>> SELECT ALL↓
```

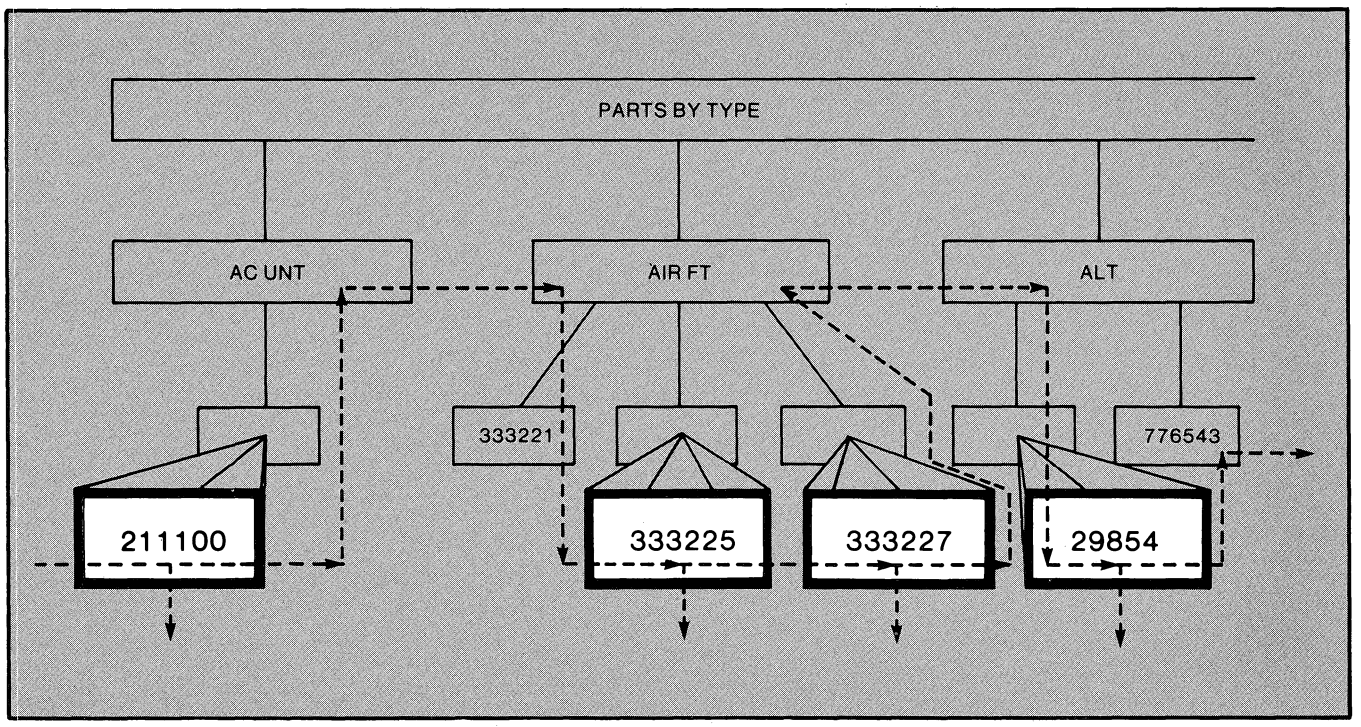
If, however, you only want to know the number of subindex records that meets the condition criteria you specified, use the /COUNT switch.

Then, the Query selects and counts the number of matching records, but will not display or output them. Instead, the system will respond with the results of the count:

n records selected

where *n* equals the number of matching records.

Your index structure affects the results of a SELECT command with a numeric argument. For instance, the index PARTS BY TYPE has a selector index above the actual data. If you travel down the index structure to the first selector, AC UNT, and ask the Query to SELECT ALL the records, it will select all the records associated with *that* selector. To select all the records from the entire subindex, you must use a /TRAVERSE switch, as shown in Figure 2-10. You don't need to use this switch in the PARTS index because there is no selector index above the actual data.



SD-02266

Figure 2-10. The SELECT/TRAVERSE Command

For example, if your path is PARTS BY TYPE:AC UNT, and you move DOWN and ask the Query to SELECT all the records that meet the condition ON HAND = 0, you won't get any records:

```
>> PATH PARTS BY TYPE:AC UNT↓
Path is PARTS BY TYPE:AC UNT
>> DOWN↓
Path is PARTS BY TYPE:AC UNT:211100
>> QFORMAT↓
The current qformat is RETAIL
>> CON ON HAND = 0↓
Condition is:     ON HAND = 0
>> SELECT ALL↓
NO records Selected
```

However, if you follow the same steps, using the /TRAVERSE switch, the Query will return seven records:

```
>> PATH↓
Path is PARTS BY TYPE:AC UNT:211100
>> QFORMAT↓
The current qformat is RETAIL
>> CON ON HAND = 0↓
Condition is:     ON HAND = 0
>> SELECT/TRAVERSE ALL↓

PART NUM      : 298543
PRICE         : 42.20
DESCRIPTION   : ALT
SUPPLIER      : RENAULT US
YEAR          : 79
MAKE          : RENAUL
MODEL         : LE CAR
ON HAND       : 0

PART NUM      : 776543
PRICE         : 42.20
DESCRIPTION   : ALT
SUPPLIER      : DATSUN INT
YEAR          : 76
MAKE          : DATSUN
MODEL         : 710
ON HAND       : 0

PART NUM      : 804560
PRICE         : 19.50
DESCRIPTION   : BRA DR
SUPPLIER      : RENAULT US
YEAR          : 78
MAKE          : RENAUL
MODEL         : LE CAR
ON HAND       : 0
```

PART NUM : 099875
PRICE : 4.88
DESCRIPTION : BRA LT
SUPPLIER : GEN MOTORS
YEAR : 74
MAKE : CHEV
MODEL : NOVA 225
ON HAND : 0

PART NUM : 056793
PRICE : 48.79
DESCRIPTION : CM
SUPPLIER : FO MO CO
YEAR : 72
MAKE : FORD
MODEL : PINTO V6
ON HAND : 0

PART NUM : 061523
PRICE : 42.00
DESCRIPTION : HTR BX
SUPPLIER : GEN MOTORS
YEAR : 74
MAKE : OLDS
MODEL : TORONAD V8
ON HAND : 0

PART NUM : 129981
PRICE : 16.23
DESCRIPTION : SE BLT
SUPPLIER : DATSUN INT
YEAR : 72
MAKE : TOYTOA
MODEL : COROLLA
ON HAND : 0

PART NUM : 767375
PRICE : 28.70
DESCRIPTION : ST WHL
SUPPLIER : DATSUN INT
YEAR : 65
MAKE : TOYTOA
MODEL : CORONA
ON HAND : 0

If you use the /RFORMAT switch, the Query uses the current rformat to produce its output. In other words, the records selected will be displayed in the form of a report. For example, if you wanted to produce a report showing the dollar value of all the air conditioning units in stock, type

```
>> RFORMAT STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
>> PATH PARTS BY TYPE:AC UNT↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
>> CONDITION DESCRIPTION="AC UNT"↵
Condition is: DESCRIPTION="AC UNT"
>> SELECT/RFORMAT ALL↵
```

The Query will display the contents of Figure 2-11 on your terminal.

As the Kid Goes for Broke
CAUCUS CAR PARTS

PAGE 1

Dollar Value of Warehouse Inventory

12/16/79

Description	*Part Number*	*Stock*	*Cost*	*Dollar Value*
AC UNT	*211100*	*1*	*79.99*	*$79.99*

Value of AC UNT Stock = $79.99

Total Value of Warehouse Stock $79.99

Figure 2-11. Using the SELECT/RFORMAT Command

If you have an rformat that defines a report that is up to 80 columns per page (see Chapter 5), you can use the /SCREEN switch. (Note that you can have only one character per column.) The /SCREEN switch uses the special features of the DASHER Display terminal to display the report screen by screen. The Report Writer will first display the report name, column heads, and any other header information. These lines will remain on your terminal screen as long as the report is being displayed. The actual data you want included in the report will fill up the screen until there is no more room. (Your terminal screen will display 22 lines.) When the 22 lines are full, the Report Writer asks if you want to see another screenful or stop. For instance, if a SELECT/RFORMAT ALL command produced a report larger than the screen could hold, the output would zip past. If you used the /SCREEN switch instead of the /RFORMAT switch, the same report appears, but only one screenful of records at a time.

You can also use the /RETURN switch with the SELECT command to keep the current path stationary when doing a multirecord SELECT.

For example:

```
>> QFORMAT PARTS↵
The current qformat is PARTS
>> CONDITION MAKE="FORD"↵
Condition is : MAKE="FORD"
>> SELECT/RETURN/L=FILENAME ALL↵
        15 records written into the list file
Path is PARTS:056793
```

Notice in the above example the use of the /L=filename switch. As explained in Chapter 1, this switch sends the output to the named file.

Up to now, we've been showing examples of the CONDITION and SELECT commands using pre-established qformats. By creating new qformats, you can broaden the range of your selection options. In Chapter 3, when we explain qformat creation, we'll talk more about record selection.

LEVELMARK Command

The index-level separating character is the *LEVELMARK*. Each time you start a Query session, the LEVELMARK character will be a colon (:). If you need to change it during your session, use the LEVELMARK command.

For example, if one of your indexes has a series of keys that contains a colon (:), you may wish to change the LEVELMARK character so you won't confuse it with the key. Type

```
>> LEVELMARK c↵
```

where c represents a single keyboard character.

The Query will respond with

Level Separator is c

If you type the LEVELMARK command without an argument, the Query will respond with the current LEVELMARK:

```
>> LEVELMARK↵
Level Separator is :
```

HELP Command

Use the HELP command whenever you need immediate and brief information about Query commands. You'll find this feature especially useful when you need advice in the midst of a Query session and you're too far away from this manual.

If you're working at the terminal and have a question about using the Query facility, simply type

```
>> HELP↵
```

The Query will respond with the contents of Figure 2-12.

Help is available on the following topics :

KEY	*QFORMS_FILE*	*CLI*	*SELECT*
PATH	*CONDITION*	*BYE*	*INDEX*
RFORMS_SYNTAX	*HELP*	*LEVELMARK*	*EX_LEVELMARK*
PRIOR	*DOWN*	*READ*	*NEXT*
UP	*EX_UP*	*EX_PRIOR*	*QFORMS_SYNTAX*
EX_DOWN	*EX_READ*	*EX_KEY*	*QFORMS_CREATING*
QFORMS_NONASCII	*QFORMS_ERRORS*	*OVERVIEW*	*EX_SELECT*
EX_PATH	*EX_CONDITION*	*RFORMAT*	*EX_RFORMAT*
EX_NEXT	*QFORMAT*	*EX_QFORMAT*	

Figure 2-12. The HELP Screen

The Query can give you general help, reference help for commands, and examples of commands.

You simply ask the Query for information on a topic by typing

>> HELP <TOPIC>↵

and the Query displays an explanation. In Figure 2-13 we show the information you would receive if you typed

>> HELP OVERVIEW↵

If you ever want help from the Query, don't be afraid to ask.

CLI Command

If at any time during your Query session you need to enter the CLI, but are not ready to end your session, you can use the CLI command. It will temporarily suspend your Query session and give control to the CLI. To enter the CLI, simply type

>> CLI↵

AOS CLI REV x.xx dd-mm-yy hr:mn:sc

)

When you're ready to return to your Query session, simply type

)BYE↵
>>

and you'll see the Query prompt (> >) back on your terminal.

Note that you cannot change @ LIST or your search list from the Query by using the CLI command. If you try to change them, when you return to the Query, you'll get the original search list and list file with which you started.

BYE Command

When you have completed your Query session use the BYE command to terminate the session and return to the CLI. Type

>> BYE↵

and the system returns a prompt

)

telling you that you're back in the CLI.

BYE is the last word you'll say to the Query.

Basically, Query allows you to navigate through an INFOS file via a PATH. Once you have established a PATH, you can READ a number of records. The PATH is a description of the current position in the INFOS file and can be changed by keyed or relative motion. The mode of printing the retrieved records depends on previously defined qformats and rformats.

There are 15 QUERY commands:

Motion	*Read/Write*	*Misc*
* *PATH*	* *READ*	* *LEVELMARK*
* *KEY*	* *CONDITION*	* *HELP*
* *NEXT*	* *SELECT*	* *CLI*
* *PRIOR*	* *QFORMAT*	* *BYE*
* *DOWN*	* *RFORMAT*	
* *UP*		

Figure 2-13. The HELP OVERVIEW Screen

RWCHECK -- The Stand-Alone Compiler

Whenever you create a new rformat, you can verify its syntax using RWCHECK, the Stand-Alone Compiler. In the next chapter we discuss creating rformats.

RWCHECK is an interactive utility that operates independently of INFOS. You don't even need an INFOS file to use it. However, you do need files containing your qformats and rformats.

RWCHECK also provides a sample page of your report if you want it. To use RWCHECK, simply type

) X RWCHECK↵

while in the CLI. The system will respond with

RFORMS Filename:

You indicate the name of your database, and the RWCHECK continues to ask you questions. The default answers appear in brackets. For instance:

RFORMS Filename: CAUCUS

Would you like to see a list of report format names? [N] Y↵
STOCK VALUE
OLDIES

Report Format Name: STOCK VALUE↵

RWCHECK then displays the entire rformat. If there are any errors, a message will appear telling you on which line the error exists. If no errors are found, RWCHECK responds with

No errors detected. Want a sample page? [Y]

If you answer Y (yes), RWCHECK asks

Name a file ...any file: [@ OUTPUT]

Indicate the name of a text file to which you want the sample page written. RWCHECK then asks

Do you wish to validate another report format? [N]

If you answer yes, the above process is repeated. If you answer no, the session ends and you'll be back in the CLI.

We'll discuss RWCHECK in more detail in Chapter 5, but for now keep in mind that it is a useful tool that tests rformats for valid syntax and semantics. In addition, RWCHECK supplies sample pages so you can see what the report will look like. This enables you to make any necessary changes to the rformat before using it to output actual data.

Summary

The Query's selection and display commands provide a versatile system to retrieve, display, and store the database records you choose to access. Thus far we've explained the various commands you need to use the Query and the Report Writer. Chapter 6 contains additional reference material about all the commands, switches, and arguments.

By now you should understand the basic principles of using the Query and Report Writer facilities. However, the best way to truly understand the Query system is to use it. If you're still unclear about the commands, or if you're uncomfortable using them, go back over this chapter and follow the examples, using the Caucus Car Parts database provided with this product.

You should understand these basic principles; they prepare you to master the more advanced features of the Query facility. In the next chapter we discuss the details you need to know to design qformats and rformats. Chapters 4, 5, and 6 contain reference material about qformats, rformats, and all the Query and Report Writer commands.

End of Chapter

Chapter 3
Tailoring the Query

As we've mentioned before, at the heart of the Query is a record selection and retrieval system. In the previous chapters we illustrated how this system works, and we showed how to use it with pre-existing qformats and rformats. This chapter describes how you can tailor the Query to fit your own INFOS system. We describe how to create, edit, and delete qformats so you can selectively retrieve and display individual record parts. We also describe how to create, edit and delete rformats so you can selectively retrieve data and display it in the form of a report. In addition, we show you how to build macros so you can repeat common query requests, how to save individual Query sessions, and how to use RWCHECK, the Stand-Alone Compiler.

Qformats

You create, edit, and store your qformats in an AOS text file called <index> .QFORMS, where <index> is the name of your current INFOS index. You can create the .QFORMS file by building a text file while in the CLI with the name < index > .QFORMS. Or if the file doesn't already exist, the Query will automatically create the .QFORMS file in your directory when you use the QFORMAT/LINEDIT or QFORMAT/SPEED command. In either case, the .QFORMS file is the working repository for your record formats.

To edit the .QFORMS file from the Query, simply type

```
>> QFORMAT/LINEDIT↵
```

or

```
>> QFORMAT/SPEED↵
```

The QFORMAT/LINEDIT or /SPEED command tells the Query to open the .QFORMS file and to call up one of the system's text editors for use. If a .QFORMS file already exists, the editor's prompt will appear on the screen, and you can begin editing the file.

NOTE: You use the same editing procedures when you use a text editor from the Query as you would from the CLI. If you are unfamiliar with the text editors, LINEDIT or SPEED, consult your *AOS Software Documentation Guide* for the appropriate manuals.

To explain the qformat creation process, let's first look at the qformats that exist for Caucus Car Parts. The .QFORMS file for Caucus Car Parts is shown in Figure 3-1. A copy of this file, called CAUCUS.QFORMS, should exist in a directory that is on your searchlist or in the current directory.

As we mentioned in Chapter 2, Caucus Car Parts maintains several qformats in the .QFORMS file. As you can see in Figure 3-1, they are PARTS, OPEN PO'S, RETAIL, and TYR PO LOCATIONS.

The PARTS qformat contains all the fields the PARTS database records contain; the OPEN PO's qformat the fields the PO records contain; and the RETAIL qformat the information needed at point of purchase. We'll talk about the purpose of the TYR PO LOCATIONS qformat later in this chapter.

Each qformat in the .QFORMS file contains three different types of lines. The first line is the *START_FORMAT* line; succeeding series are the *FIELD DESCRIPTOR* lines; and the last is the *END_FORMAT* line.

In addition, as you'll notice in Figure 3-1, each FIELD DESCRIPTOR line consists of the name of the field and two numbers. These numbers indicate the starting and ending byte of each field. For instance, in the qformat RETAIL, the field called *PART NUM* starts at byte 1 and ends at byte 6. We'll talk more about this later.

Earlier, we demonstrated how you use the qformats to specify CONDITION statements that SELECT certain records conforming with established criteria. For many of your queries, the qformats that describe your entire database record will be sufficient.

```
START_FORMAT      RETAIL
PART NUM          1     6
PRICE             60    65
DESCRIPTION       7     12
SUPPLIER          40    49
YEAR              13    14
MAKE              15    20
MODEL             21    30
ON HAND           31    33
END_FORMAT
START_FORMAT      PARTS
PART NUM          1     6
DESCRIPTION       7     12
YEAR              13    14
MAKE              15    20
MODEL             21    30
ON HAND           31    33
IN WAREHOUSE      34    36
REORDER           37    39
SUPPLIER          40    49
SUP ADDRESS       50    59
RETAIL PRICE      60    65
WHOLESALE PRICE   66    71
END_FORMAT
START_FORMAT      OPEN PO'S
PO NUM            1     6
DESCRIPTION       7     12
YEAR              13    14
MAKE              15    20
MODEL             21    30
PART NUM          31    36
DATE              37    42
QTY               43    44
RETAIL PRICE      45    50
WHOLESALE PRICE   51    56
SUPPLIER          57    66
SUP ADDRESS       67    76
END_FORMAT
START_FORMAT      TYR PO LOCATIONS
SHORT DES         10    12
SUPPLIER          57    66
SUP LOCATION      75    76
END_FORMAT
```

Figure 3-1. The .QFORMS File for Caucus Car Parts

However, using the full record format presents two serious limitations. First, you can only search for the contents of a given record field as it appears in full record format; you can't search for separate pieces of a field. Second, the Query displays the entire record when it finds one that meets the specified conditions.

Oftentimes, though, you'll want to search for parts of a field, and display only those parts of the full record. To perform these selective procedures, you will need to create additional qformats that isolate those fields you want to access and mask the unwanted ones.

Understanding Record Structures

You can use the Query to create various qformats for your own database records. In fact, the number and variety of qformats you create improves the Query's flexible retrieval power.

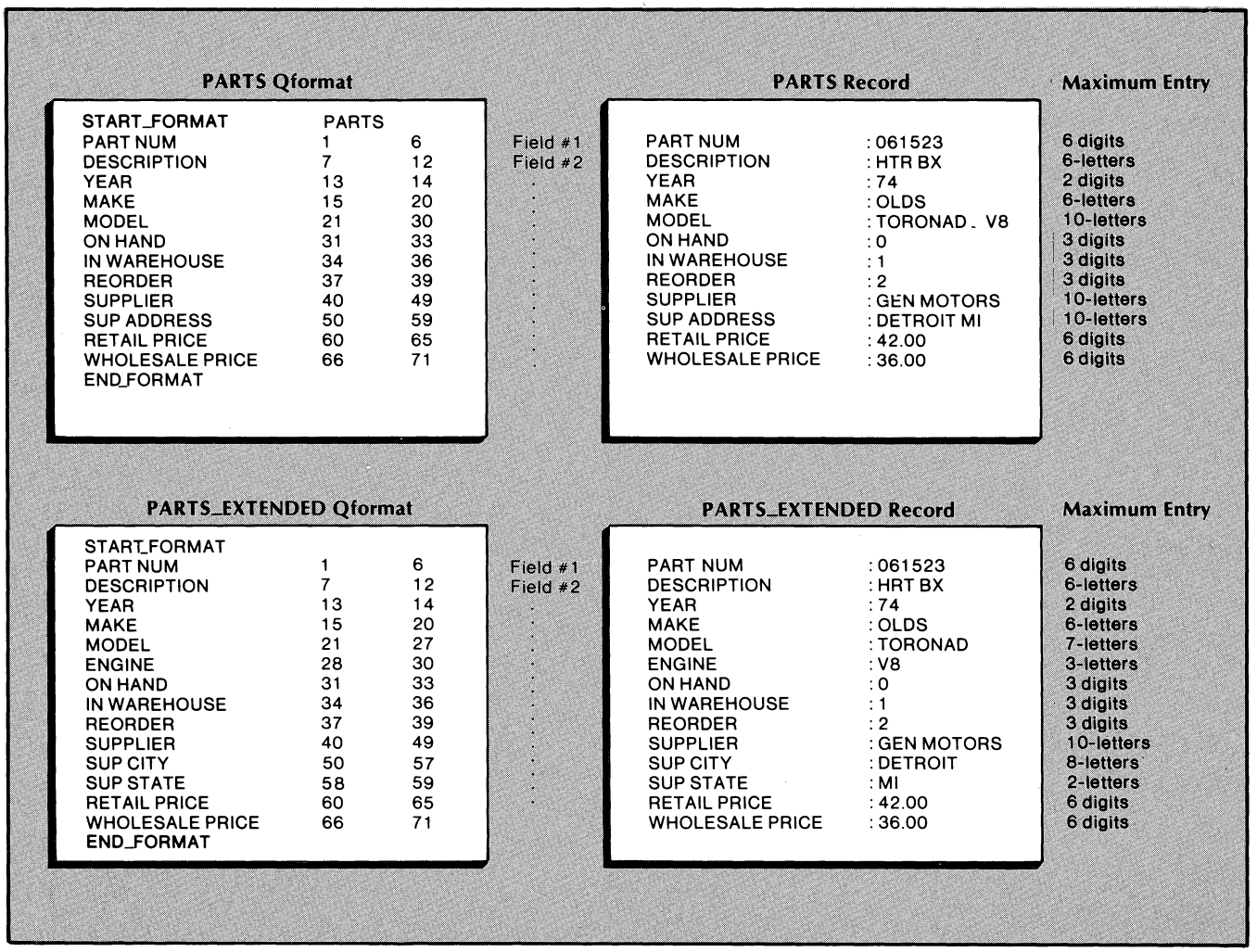
Depending upon the method used to create the database records, you can feature any or all bytes in the record format. Therefore, you should have an understanding of the internal structure of the records contained in your database.

The Caucus database, for example, contains records with the following internal structure (see Figure 3-2).

1. The field *SUP ADDRESS* contains standard abbreviations for the city and the state. Each abbreviation is contained in specific bytes of the field, so Caucus can isolate only the state abbreviation in a qformat.

2. The fields for make and model contain certain standard maximum abbreviations. Also, the field for model has the engine displacement in a separate range for easy retrieval.

3. The date field can be broken down into three individual ranges, allowing you to separately access the month, the day, or the year.



SD-02268

Figure 3-2. The Internal Structure of Caucus Records

As a general rule, the more internally structured you create your records, the more specific and flexible you can build the qformats. For example, Table 3-1 describes the internal structuring of a typical personnel record.

Using this record as a model, you can imagine the variety and complexity of qformats and CONDITION statements that you can create. Table 3-2 contains a partial list of the possibilities.

Creating qformats

To create a new qformat, you simply list the information you want the Query's "eyes" to "see" and its location within the record. For example, Caucus Car Parts wanted a qformat that would enable them to select all the PO records containing tires on order from suppliers in Michigan. The two necessary categories of information are

1. the part description, and
2. the supplier's name, address, and state.

Now if you were working at Caucus and tried to select these records using only the full record format -- OPEN PO's -- you would have to specify a different CONDITION statement for each type of tire and each city in Michigan where a supplier was located. That would be a major task. Instead, you could take advantage of the fact that the database contains a finer structure; you could create a separate qformat (as they did at Caucus) called *TYR PO LOCATIONS* that would use the following:

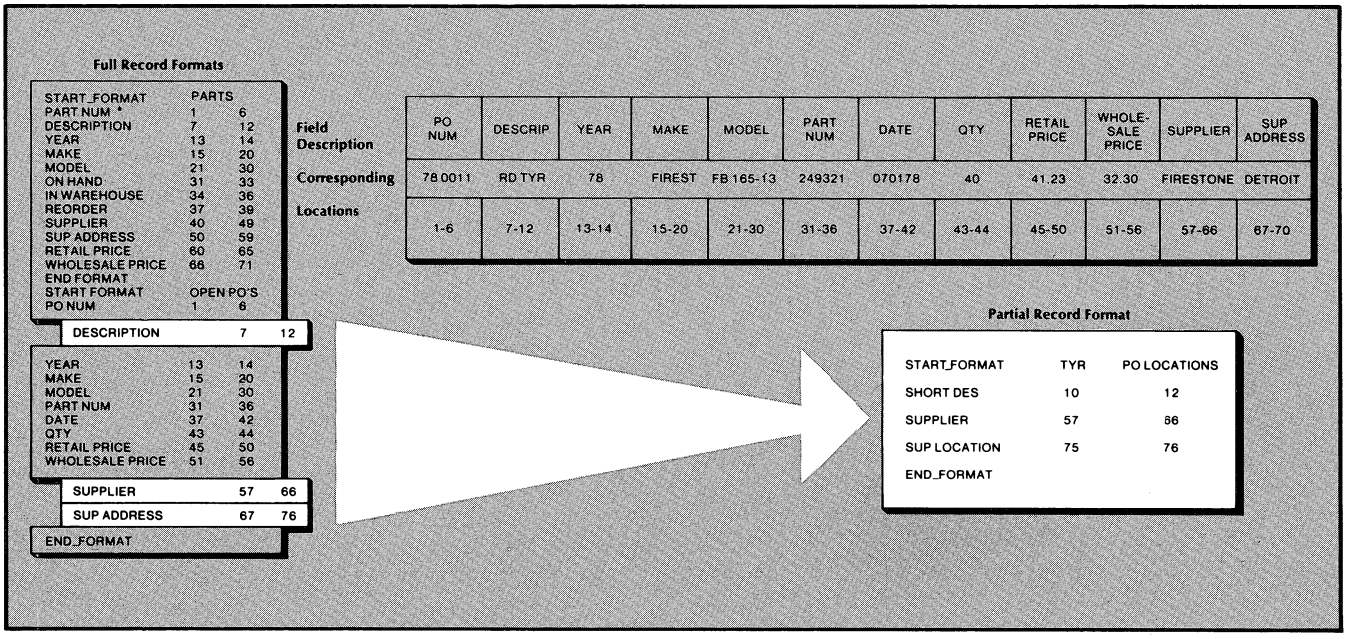
1. the second part of the field *DESCRIPTION*, renamed *SHORT_DES*, which would contain *TYR;*
2. the field *SUPPLIER;*
3. the last part of the field *SUP ADDRESS*, renamed *SUP LOCATIONS*, which would contain the state or country. Figure 3-3 shows how this qformat would be taken from the full PO record format.

Table 3-1. The Structure of a Typical Personnel Record

DATANAME	BYTES				
NAME	1-4	5678	9-13		
	RUSS		GREGO		
	Last Name		First Name		
ADDRESS	14-16	17-25	26-29	30-31	
	010	DOWNINGST	WORC	MA	
	St.□	St. Name	City Abbrev.	State Abbrev.	
POSITION	32-25	36-37			
	MNGR	10			
	Position	Years in Position			
DEPT	38-44	45-46	47-50		
	PROGRAM	RD	WEST		
	Dept. Abbr.	Research Development	Location		
SALARY	51-55	56-57			
	32 000	DX			
	Salary in Dollars	Salary Code			
DEDUCTIONS	58-63	64-69	70-74	75-79	80
	252.00	101.00	18.20	19.00	A
	Fed. Tax	State Tax	FICA	Stocks	Weekly = A Bi-wkly = B

Table 3-2. A List of Query Possibilities

Purpose	Format Condition Statement			Result
	Dataname	Bytes	Condition	
What employees have been at their position for more than 5 years.	EMP NAME POSITION	1-13 36-37	POSITION > "05"	Query returns all employees at their positions for more than 5 years.
Which department managers earn the highest salary?	EMP NAME POSITION SALARY DEPT	1-13 32-35 51-55 45-46	POSITION= "MNGR" /AND DEPT = "RD"	Query returns all DEP. MANAGERS with their current salaries.
Which employees spend more on the stock plan than they pay in state tax?	EMP NAME STOCK ST_TAX	1-13 75-79 64-69	ST_TAX<STOCK	Query returns all employees who spend more on stocks than paid in state tax.
Which manufacturing employees live in Boston?	EMP NAME ADDRESS DEPT	1-13 26-29 45-46	ADDRESS= "BOST" /AND DEPT = "MF"	Query returns all manufacturing employees living in BOSTON.



SD-01341

Figure 3-3. Full and Partial Record Formats

As you can see, each Purchase Order record in the database contains 76 bytes divided into twelve fields. To create a new qformat, like TYR PO LOCATIONS, you must follow three steps:

First, open the .QFORMS file using the QFORMAT/LINEDIT or /SPEED command. Or, you may open the .QFORMS file from the CLI as we mentioned before.

Second, you would append the new qformat to the file. To create the first line of the new qformat, type the keyword START_FORMAT followed by a tab, followed by the name of your new qformat. In this example, the qformat name is TYR PO LOCATIONS.

The second line of the qformat begins a listing of the individual fields that you want the qformat to contain. You can list these fields in any order; the Query does not require a particular order.

Continuing with the example in Figure 3-3, you might begin your list of relevant fields with the field *SHORT DES* (i.e., a shorter description) followed by a tab, followed by the start byte of the address of the description.

As you can see from Figure 3-3, the start byte of the field *DESCRIPTION* in the PO records is 7, so you would type 7 for the start byte position, followed by a tab, followed by the end-byte position. Here's where you need to be careful.

We mentioned before that you want to feature only the second part of the field DESCRIPTION, masking the other parts in your search, thereby isolating the abbreviation *TYR*. Therefore, instead of using 7 as the start-byte position, type 10 followed by a tab, followed by the end-byte position of 12. In this way, the Query will only "see" the last three bytes in the DESCRIPTION field, the bytes that contain TYR. After the end-byte, you may type a tab, followed by an optional datatype string that changes the datatype of the field, if you want the field displayed in a mode other than ASCII. If you don't specify a datatype string, the field will be displayed in ASCII characters by default.

You follow this procedure, listing the fields and their corresponding byte locations, for all the fields you want to include in your qformat. You must be sure to delimit each entry with a tab. If you don't, the Query will not accept the qformat and will send you an error message upon re-entry to the Query session.

The next field you might include in this example could be *SUPPLIER*. On the next line, type SUPPLIER followed by a tab, followed by the start-byte of the supplier's name, which is 57.

Notice that you should include the entire range for SUPPLIER; type 57 for the start-byte, followed by a tab, followed by the full range end-byte 66. Since you don't need to specify a datatype other than ASCII, you can leave the last category blank.

To include a field in the qformat that would allow you to select a particular state (or country), you would only specify a section of the field *SUP ADDRESS*. In Figure 3-3, the field SUP ADDRESS contains two pieces of information: the city and the state (or country). Since the state is the second part of the field, you would specify a field called *SUP LOCATIONS,* including a start-byte of 75, followed by a tab, followed by end-byte of 76. This two byte range would then cover only the abbreviation for the state (or country), masking the city.

Last, specify the keyword END_FORMAT to tell the Query to collate all of the previous field description lines into a single record format.

If you edited the .QFORMS file from within the Query, when you leave the editor, the Query will check for errors in the qformat. It will automatically check to make sure that the start-bytes and end-bytes on the field descriptor lines are numbers and that the start-byte is less than or equal to the end-byte. The Query also checks to make sure that you have used the correct syntax in the qformat. If you have made any errors in creating your qformat, you'll receive one of the following error messages:

START_FORMAT must have a qformat name - on line x
Looking for a START_FORMAT on line x
Bad START BYTE on line x
Bad END BYTE on line x
Bad KEY TYPE on line x
START BYTE is greater than END BYTE on line x

These errors will be followed by the phrase

x is not a valid qformat

If you created your new qformat outside the Query in the CLI, you'll receive any error messages at the beginning of the Query session and you will not be able to access those qformats containing errors.

When a qformat is obsolete, you can delete it by erasing the respective lines in the .QFORMS file with a text editor. When you delete an existing qformat, the Query will automatically erase the qformat's name so it will not appear if you use the QFORMAT/NAMES command.

Using Your New Qformat

Using the new qformat TYR_PO_LOCATIONS, you can now declare a condition statement with the following criteria:

```
CONDITION SHORT DES = "TYR"
CONDITION/AND SUP LOCATION= "MI"
```

Now, if you go to the PO index, the Query will select only those records that contain the specified conditions.

For instance, once the new qformat has been appended to the .QFORMS file, you can activate the qformat and conditions:

```
>> QFORMAT TYR PO LOCATIONS↵
The current qformat is TYR PO LOCATIONS
>> CONDITION SHORT DES = "TYR"↵
CONDITION IS: SHORT DES = "TYR"
>> CONDITION/AND SUP LOCATION = "MI"↵
CONDITION IS: SHORT DES = "TYR"
        AND   SUP LOCATION = "MI"
```

You move to the PO subindex and tell the Query to search through the subindex for matching records:

```
>> PATH PO↵
Path is PO
>> DOWN↵
Path is PO:770945
>> SELECT ALL↵
SHORT DES       :TYR
SUPPLIER        :FIRESTONE
SUP LOCATION    :MI

SHORT DES       :TYR
SUPPLIER        :GOODYEAR
SUP LOCATION    :MI
>> BYE↵
```

Rformats

You create, edit, and store rformats in an AOS text file called < index > .RFORMS, where < index > is the name of your current INFOS index. As with the .QFORMS file, you can create the .RFORMS file while in the CLI by building a text file with the name <index> .RFORMS. Or, if the file doesn't already exist, the Query will automatically create the .RFORMS file in your directory when you use the RFORMAT/LINEDIT or RFORMAT/SPEED command. In either case, the .RFORMS file is the working repository for your rformats.

You follow a few simple steps when creating rformats:

- Use a text editor to create and/or edit the .RFORMS file.
- Use RWCHECK to verify the syntax.
- Examine the sample page to see what the report will look like.
- Repeat the previous steps until satisfied.
- Use the Query to produce a report with real data.
- Further modify the report, if necessary, by invoking a text editor from within the Query or from the CLI.

To edit the .RFORMS file from the Query, simply type:

```
>> RFORMAT/LINEDIT↵
```

or

```
>> RFORMAT/SPEED↵
```

The RFORMAT/LINEDIT or /SPEED command tells the Query to open the .RFORMS file and to call up the appropriate text editor. If a .RFORMS file already exists, the editor's prompt will appear on the screen, and you can begin editing the file.

NOTE: You use the same editing procedures when you use a text editor from the Query as you would from the CLI. If you are unfamiliar with the text editors, LINEDIT or SPEED, consult your *AOS Software Documentation Guide* for the appropriate manuals.

As you'll notice, the initial steps of the procedure for creating rformats is similar to the procedure for creating qformats. Keep in mind, however, that when you edit rformats from within the Query, you use the RFORMAT command.

Let's look at the rformats that exist for Caucus Car Parts. The .RFORMS file for Caucus Car Parts is shown in Figure 3-4.

Caucus Car Parts has two rformats: *STOCK VALUE* produces a report with the total value of the merchandise currently stored in the Caucus warehouse, and *OLDIES* lets management determine the revenue generated by returning old parts.

```
START_REPORT        STOCK VALUE
HEADER    1         C        "As the Kid Goes for Broke"
!
!                   This produces a report with the total value of the current
!                   warehouse stock
!
QFORMAT             PARTS
LN/PG               60
COL/LIN             80

DEFINE    DOLLAR VALUE   IN WAREHOUSE * WHOLESALE PRICE
PICTURE   DOLLAR VALUE   $$$$$99V.99

HEADER    2         C        "CAUCUS CAR PARTS"
HEADER    2         70       PAGE
HEADER    4         C        "Dollar Value of Warehouse Inventory"
HEADER    4         70       DATE

HEADER    6         5        "Description"
HEADER    7         5        "**********"

DETAIL    1         5        DESCRIPTION
BREAK     DESCRIPTION        NO_REP
BREAK     DESCRIPTION        POST_BREAK_SPACE
BREAK     DESCRIPTION        40    "Value of"
BREAK     DESCRIPTION        49    DESCRIPTION
BREAK     DESCRIPTION        56    "Stock ="
BREAK     DESCRIPTION        66    TOTAL ( DOLLAR VALUE )

HEADER    6         20       "Part Number"
HEADER    7         20       "**********"
DETAIL    1         20       PART NUM

HEADER    6         35       "Stock"
HEADER    7         35       "*****"
DETAIL    1         35       IN WAREHOUSE

HEADER    6         45       "Cost"
HEADER    7         45       "******"
DETAIL    1         45       WHOLESALE PRICE

HEADER    6         60       "Dollar Value"
HEADER    7         60       "**********"
DETAIL    1         60       DOLLAR VALUE

TOTAL     3         30       "Total Value of Warehouse Stock"
TOTAL     3         65       TOTAL ( DOLLAR VALUE )

END_REPORT
```

Figure 3-4. The .RFORMS File at Caucus Car Parts (continues)

```
START_REPORT      OLDIES
!                 A REPORT FOR MANAGEMENT TELLING HOW MUCH REVENUE
!                 CAN BE GENERATED BY RETURNING OLD PARTS
HEADER    1       C       "CAUCUS CAR PARTS"
HEADER    2       C       "Return Value of Old Inventory"
HEADER    3       60      PAGE
HEADER    4       60      DATE

QFORMAT   PARTS
LIN/PG    30
COL/LIN   80

DEFINE    DOLLAR  VALUE   ON HAND * WHOLESALE PRICE
DEFINE    RETURN  VALUE   90 % DOLLAR VALUE
PICTURE   RETURN  VALUE   $$$$$99V.99

SORT      YEAR

BREAK     YEAR    PRE_BREAK_SPACE
BREAK     YEAR    PRE_BREAK_SPACE
BREAK     YEAR    PRE_BREAK_SPACE
BREAK     YEAR    21      "Return Value of"
BREAK     YEAR    38      YEAR
BREAK     YEAR    41      "Stock = "
BREAK     YEAR    52      TOTAL ( RETURN VALUE )

END_REPORT
```

Figure 3-4. The .RFORMS File at Caucus Car Parts (concluded)

Format Descriptor Lines

As you'll notice, an rformat always begins with a START_REPORT line and ends with an END_REPORT line. It's similar to the qformat which always begins with a START_FORMAT line and always ends with an END_FORMAT line. But that's where the similarity ends. While the qformat has one type of field descriptor lines, the rformat consists of a variety of lines. These format descriptor lines may appear anywhere between the start and the end of the rformat.

The format descriptor lines are

Comment line	DETAIL lines
QFORMAT line	PICTURE lines
LIN/PG line	SORT lines
COL/LIN line	BREAK lines
HEADER lines	TOTAL lines
DEFINE lines	

Some of these lines only appear once per rformat; others may appear many times, as we'll see later on. In fact, the bare minimum an rformat can include, in addition to the START_ and END_REPORT lines, is at least either one DETAIL line, one BREAK line, or one TOTAL line.

To understand the importance of these different lines, let's first understand how the Report Writer uses the rformat. The Report Writer reads through the rformat, making note of all the different command lines and the different actions each will require, namely:

- HEADER lines tell the Report Writer that header information needs to be displayed at the top of each page of the report.
- DETAIL lines tell the Report Writer what information from each record read must be displayed.

- TOTAL lines tell the Report Writer what computations and summary information must appear at the end of the report.

- BREAK lines tell the Report Writer to suspend displaying DETAIL information when a specific data value changes, perform some other type of action, then continue displaying DETAIL information.

The other format descriptor lines don't cause the Report Writer to take specific actions, as we'll see later on. They tell the Report Writer how you want the report to look.

Let's say you've defined an rformat that consists of HEADER, DETAIL, and TOTAL lines. You call up the Report Writer during a Query session (by using a /RFORMAT switch with a READ or a SELECT command). The Report Writer reads through the rformat and knows that you want it to display header, detail and total information. It first displays the header information at the top of the first page. It then reads one record at a time and, for each record, displays the appropriate detail information. Since this report exceeds one page, the Report Writer again displays the header information at the top of the second page. This process continues until it reads all the records. The Report Writer then performs any computations needed for the total information, and displays this information at the end of the report.

If your rformat includes BREAK lines, when a specific data value changes (as defined in the BREAK line), the Report Writer stops displaying detail information long enough to execute the actions specified by the BREAK line. The Report Writer then continues to display detail information until it has read all the records. The Report Writer again performs any BREAK action before displaying the total information.

We introduce all the format descriptor lines in the following section. Again, Chapter 5 contains more detailed reference material about these lines.

Comment Line

The person who designs an rformat may not be the only one who uses it. Therefore, it's handy to document the rformat with a comment line.

As you can see in Figure 3-4, the creator of the STOCK VALUE rformat wrote the following comment:

```
!       This produces a report with the total
!       value of the current warehouse stock
```

If you were working at Caucus Car Parts and wanted to know the value of the current warehouse stock, you would find out a rformat already exists for this purpose by reading the comments in the.RFORMS file. Even if you didn't know what the other lines in the .RFORMS file meant, you'd be able to understand the purpose of this rformat.

QFORMAT Line

The Report Writer uses the same .QFORMS file as the Query. When you create an rformat, you specify an already defined qformat for your report in the QFORMAT line. In other words, the report will use certain data items from the database. Because you want to be able to refer to them by name you choose a qformat that has specified those data items.

Both STOCK VALUE and OLDIES use the qformat PARTS.

LIN/PG and COL/LIN Lines

The Report Writer gives you the flexibility of designing the size and shape of your report; your goal is a readable, well organized report.

You use the LIN/PG line to tell the Report Writer how many lines to print on each page. The default is 60, the minimum 30.

The COL/LIN line tells the Report Writer how wide a report you want. The default is 80. Note that 80 columns per line is the same as 80 characters per line; you are allowed one character per column.

If you want to use the /SCREEN switch in conjunction with the /RFORMAT switch on a READ or SELECT command, you must define your rformat with no more than 80 characters per line.

HEADER Lines

To make your report more readable, you can specify header information and column headers. If you're going to have a column of stock numbers, you can label it as such.

You'll generally have several HEADER lines in an rformat. You don't have to define them in any special order; the Report Writer checks that items on the same HEADER line do not overlap. For instance, in STOCK VALUE, HEADER line 6 defines several different headers. When the report is produced, these different items will appear across the page on the same line. In addition, the Report Writer displays header information on each page of the report; if you have a three page report, the Report Writer will print the header information three times -- once on each page.

You define the actual words you want to appear as header information as a literal or data item in the rformat (see Chapter 5 for details). If you use a literal, you must enclose it in quotation marks. Be sure to match these quotation marks. If you use a double quotation mark (") on one side, use it on the other; likewise, if you use a single quotation mark (') on one end, use a single one on the other.

DEFINE Lines

You use DEFINE lines if you want additional data items to appear on the report that do not appear in the record. DEFINE lines define temporary variables that are considered part of the records for the duration of the Report Writer process. These additional data items are the result of computations done with other data items in the record.

For instance, if you want to print the result of multiplying two data items, use a DEFINE line to create a data item to hold the result. This data item will only exist for the duration of the Report Writer process.

In STOCK VALUE, the DEFINE DOLLAR VALUE line creates a temporary data item to hold the result of multiplying the IN WAREHOUSE and WHOLESALE PRICE data items.

DETAIL Lines

You use DETAIL lines to specify the data you want included in your report, for each input record. You can specify either data items or literals in the DETAIL lines. If you specify a literal, you must include it in quotation marks. Be sure the quotation marks match on both sides of the literal. If you use a double quotation mark on one side, use it on the other; if you use a single quotation mark on one side, use a single one on the other.

In STOCK VALUE, the DETAIL lines include the PART NUM, IN WAREHOUSE, WHOLESALE PRICE, and DOLLAR VALUE data items.

PICTURE Lines

A PICTURE line describes how you want a particular numeric item to appear on the printed report. If you do not define a PICTURE line, the Report Writer outputs any numeric data in a 14 character field with two decimal places.

The PICTURE line allows you to define the output using a subset of the PL/1 and COBOL picture facilities. The Report Writer recognizes seven symbols: 9, $, V, S, /, .(period), and ,(comma). These symbols are explained in Chapter 5.

In STOCK VALUE, a PICTURE line describes DOLLAR VALUE so it will appear with a dollar sign and a decimal point on the finished report.

SORT Lines

You use SORT lines if you want the records to appear in the report in a special order. You can specify that the records appear in ascending, or descending order according to a specific key. You can have multiple SORT lines in an rformat; if there is more than one, the keys stipulating the sorting order must appear in the correct order in the rformat.

Note, however, that if your records are already sorted in a particular order in the database, a SORT line is unnecessary, inefficient, and may be time consuming.

BREAK Lines

A BREAK line tells the Report Writer to take specific action when the value of a data item changes. For instance, in STOCK VALUE, the BREAK lines tell the Report Writer to skip lines and print a new message when the DESCRIPTION data item changes. In other words, the Report Writer is instructed to follow a certain steps until it encounters a different DESCRIPTION.

You can use BREAK lines to instruct the Report Writer to

- print a line containing summary information,
- suppress repeated fields in DETAIL lines,
- skip to a new page or a new line,
- underline specific columns.

TOTAL Lines

You can tell the Report Writer to print summary information at the end of the report with a TOTAL line. That is, the TOTAL command line also instructs the Report Writer to perform calculations, and then use the results as the summary information.

In STOCK VALUE, TOTAL lines print the caption "Total Value of Warehouse Stock," and then the actual total as computed by the Report Writer.

General Rules for Defining Rformats

We've just given you a brief summary of the different lines you must include in an rformat; Chapter 5 contains more details. In the next section we'll offer some suggestions that you might find helpful when defining rformats. What you do with the various format descriptor lines, and how you use them to define your rformats depends on the report you want to design. The ultimate report is something only you can determine.

However, there are some general rules you should keep in mind:

1. The rformat must always begin with a START_REPORT line and end with an END_REPORT line.

2. The QFORMAT line must precede any line in the rformat that refers to a data item.

3. You must use a DEFINE line to define a data item before you can use that data item in another descriptor line. We refer to a defined data item as a def_item (defined data item).

4. You may only define a data item once per rformat.

5. You may only use a PICTURE command line once per data item.

6. You cannot have multiple COL/LIN or LIN/PAGE lines in an rformat.

7. A DETAIL line for an data item must precede any BREAK line that calls for underlining that data item.

The order of the different format descriptor lines is flexible. You can group these lines in your rformat so they are easy for you to read and understand. No matter how you use the various format descriptor lines, the Report Writer checks that you have followed the rules.

Using RWCHECK

RWCHECK, the Stand-Alone Compiler, is an interactive utility that verifies the syntax of an rformat. You can also use this useful tool to provide sample pages of your report.

You invoke RWCHECK from the CLI using the following command:

) XEQ RWCHECK *[name[.RFORMS]]* ⤸

where *name.RFORMS* is the file containing your rformats.

If you want, you can use name.RFORMS as an argument; if you choose not to, RWCHECK asks for a file name as the first question in the ensuing dialog. RWCHECK then asks which rformat you want verified, whether or not you want a sample page, and where you want the sample page sent. This can be very valuable when first defining rformats; before actually using the rformat to display data, you are able to see how the finished report will look.

The actual dialog differs depending on the switch(es) you use when invoking the RWCHECK (see Chapter 5). RWCHECK can also operate in batch mode, if invoked with the appropriate switch.

In Chapter 5 we show a sample dialog of the RWHCECK session and some sample pages.

Suggestions for Designing Reports

The reports you ultimately generate depend on the type of data in your database and how you use this data. As long as you follow the few simple, flexible syntax rules, you can design a variety of different reports.

It isn't feasible for us to show you every possible rformat variation; there are too many possibilities and your needs are so individualized. All we can offer is a few hints and suggestions to consider when designing your reports.

Since you use the system's text editors to create rformats, you can always make a change. In fact, we suggest that you use RWCHECK to produce a sample page before using the rformat in an actual application. That way, you'll see what the report will look like before there's data in it. For instance, if you want to change the header information after seeing it on the sample page, simply call up the text editor again and make the necessary changes. You don't have to use the rformat until it produces the exact report you want.

The Report Writer automatically inserts a blank line between header information and detail information on the displayed report. If you define more than one line of header information and do not specify consecutive line numbers in the HEADER command lines, the Report Writer will insert blank lines between them. In other words, if you want header information to appear on lines 1, 4, and 6, the Report Writer will automatically leave lines 2, 3, and 5 blank.

Keep in mind the difference between the rformat and the report. The rformat defines the report, and ultimately you want a readable report. You can achieve this and still have a readable rformat. For instance, you don't have to group all HEADER lines together in the rformat. The Report Writer will find them all, no matter where they appear in the rformat. Therefore, it might make a more readable rformat if you intersperse HEADER lines with DETAIL lines. And if blank lines every now and then in your rformat makes it easier to read, put them in.

For example, your rformat could include the following descriptor lines:

```
HEADER 6      5     "Description"
HEADER 6      20    "Part Number"
HEADER 7      5     "**********"
HEADER 7      20    "**********"
DETAIL 1      5     DESCRIPTION
DETAIL 1      20    PART NUM
```

However, the report would look exactly the same if you used these same descriptor lines, but ordered them as follows:

```
HEADER 6      5     "Description"
HEADER 7      5     "**********"
DETAIL 1      5     DESCRIPTION

HEADER 6      20    "Part Number"
HEADER 7      20    "**********"
DETAIL 1      20    PART NUM
```

Although both define header information that includes two column titles underlined with asterisks and two columns of detail information from data_items DESCRIPTION and PART NUM, the second example makes an easier to read rformat.

Because individual data processing needs will dictate the types of reports you design, it's impossible for us to anticipate all the different types you might need. However, keep in mind that rformats are flexible and the Report Writer will check for any sytnax errors. As we've said before, the best way to understand the entire process is to do it yourself.

Chapter 5 contains the syntactical parameters you need for each format descriptor line. We suggest you review them before defining any rformats.

Using Rformats

Once you've defined rformats that produce the reports you need, you can use the Query to put data into the reports. As mentioned in Chapter 2, if you use the /RFORMAT switch with a READ or SELECT command, the data will be displayed according to the currently active rformat. These reports can either appear on your terminal or go to a text file that you can print later.

In addition, if you find that you often need to create the same report, you can include the appropriate READ and/or SELECT comand, with the /RFORMAT switch, in a Query macro, as explained in the next section.

Query Macros

After working with the Query facility a while, it's possible that you may need many of your queries on a regular basis. Instead of repeatedly typing the same condition statements to access your qformats, or using the same READ command to create reports, you can save individual Query sessions for later use by building Query macros.

Building Macros

The term macro originates from the Greek term meaning large or complex, so think of Query macros as large, complex commands. Essentially, a Query macro is a text file in your directory containing a series of commands that will easily accomplish a difficult or time-consuming task. The macros you build for the Query can be of varied length and complexity. In this section, we describe various methods and quidelines for building macros, and we give you suggestions for listing them.

There are two ways to build macros:

1. Create a text file in CLI that contains the commands you want the macro to perform, or

2. Use the */Q=filename* switch when invoking the Query from the CLI to save the commands used during the Query session.

In either case, you'll end up with a text file that holds the commands you want the macro to perform. You can edit this text file at anytime from the CLI.

Macro Building from the CLI

If you want to build a macro while in the CLI, simply invoke a text editor and create the file. This can be done before, after or during a Query session. If done during a Query session, use the CLI command to temporarily execute the CLI.

For instance, if you're in the middle of a Query session and decide that you want to create a macro called FORD_DIS_CP.QU, type

```
>> CLI↵

AOS CLI REV x.xx      dd-mm-yy      hr:mn:sc

) X LINEDIT FORD_DIS_CP.QU↵
DO YOU WANT THIS FILE CREATED? Y↵

?
```

Once in the text editor, type the commands that you want the macro to perform. When you're through, you can return to the Query, ready to try out your macro. To run the macro, you simply type its name, with or without the .QU extension (the name of the text file containing the macro always includes a .QU extension if the macro was created with a /Q=filename switch, as explained in the next section).

At any later time, if you decide to change the macro, you can again invoke the text editor and edit the macro file.

Saving Query Sessions

You can also build a macro file by saving the commands you used in a Query session. If you know that the Query session is one that you want to save, you can log on to the Query using the /Q=filename switch. The system creates and/or appends to a text file when you invoke the Query, and saves all the commands you issue during the Query session in this text file. You name the file at the same time you invoke the Query. For example:

```
) Query/Q=TEST1.QU CAUCUS↓
INFOS Query / Report Writer .... At Your Service
>> PATH↓
Path is above the index
>> DOWN↓
Path is PARTS
>> DOWN↓
Path is PARTS:056793
>> QFORMAT PARTS↓
The current qformat is PARTS
>> READ↓

PART NUM          : 056793
DESCRIPTION       : CM
YEAR              : 72
MAKE              : FORD
MODEL             : PINTO V6
ON HAND           : 0
IN WAREHOUSE      : 0
REORDER           : 1
SUPPLIER          : FO MO CO
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 48.79
WHOLESALE PRICE   : 39.50
>> BYE↓

)
```

The file TEST1.QU now contains a list of your Query commands, which you can verify by typing the file:

```
) TYPE TEST1.QU↓

PATH
DOWN
DOWN
QFORMAT PARTS
READ
BYE
```

When you're back in the Query and want to run this macro, you simply type TEST1.QU and the Query executes the macro:

```
>> TEST1.QU↓
> > PATH
Path is above the index
> > DOWN
Path is PARTS
> > DOWN
Path is PARTS:056793
> > QFORMAT PARTS
The current qformat is PARTS
> > READ

PART NUM          : 056793
DESCRIPTION       : CM
YEAR              : 72
MAKE              : FORD
MODEL             : PINTO V6
ON HAND           : 0
IN WAREHOUSE      : 0
REORDER           : 1
SUPPLIER          : FO MO CO
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 48.79
WHOLESALE PRICE   : 39.50
> > BYE
)
```

Notice the BYE command is the last line in the macro. When the Query reached it, your session ended. If you don't want the end of the macro to be the end of your Query session, you should delete the BYE command from the macro file.

In some cases, however, you may want the BYE command to remain as part of the macro file. The file you created with the /Q switch can be used to run an entire Query session under batch mode.

To run a Query session under batch mode, you use a /B switch when you initially invoke the Query. When you use the /B, you also use the name of the file previously created with the /Q and the Query executes its contents as a noninteractive session.

For example, if you type

```
) QUERY/B=TEST1.QU CAUCUS↵
```

The Query executes the contents of the /Q file, TEST1.QU, under batch mode. The Query session echoes on your terminal, but you cannot enter any commands. A BYE command must be the last command in the macro file if you plan to run it in batch mode.

If you use the AOS batch mode, the session is sent to the batch output file, rather than your console. To use AOS batch, type the following from the CLI:

```
) QBATCH QUERY/B=TEST1.QU CAUCUS↵
```

For more information about running in AOS batch mode, see the *AOS Command Line Interpreter (CLI) User's Manual.*

When you want to build macros that are not transcripts of a previous Query session, you use one of the two system text editors, LINEDIT or SPEED, from the CLI.

For example, while in the CLI you execute a text editor, which in turn asks if you want to create the specified file:

```
) XEQ LINEDIT OLD_PARTS.QU↵
DO YOU WANT THIS FILE TO BE CREATED? Y↵
```

You enter the commands you want the macro to perform:

```
?INSERT↵
PATH PARTS↵
DOWN↵
QFORMAT PARTS↵
CONDITION YEAR < 68↵
?BYE↵
)
```

The macro OLD_PARTS.QU now exists in your directory.

You can edit and delete macros in the same way that you edit and delete other files. Using one of the text editors, you can keep the macros in your directory up to date and ready to use.

Tips for Building Macros

You should follow several guidelines when you build macros:

1. Make sure the macros are self-contained. You should be able to execute them anywhere in your index. In addition, any Query actions performed before or after executing the macro are unaffected and separate from the macro actions.

2. Macros don't require any user input. Therefore, don't include any commands that require additional input, such as the CLI, the QFORMAT/LINEDIT or /SPEED, or RFORMAT/LINEDIT or /SPEED commands.

3. You can nest one or more macros within a macro; the Query will then execute the contents of any embedded macros as it would execute individual commands.

4. Keep in mind that after you execute a macro, the QFORMATS, RFORMATS, CONDITIONS, and PATHs that the macro establishes remain current in the Query. If you continue working without changing the status of these specifications, you may retrieve the wrong records, or you may receive an incorrect response from a SELECT command.

Self-contained Macros

Unless a macro is self-contained, you can't execute it anywhere in your index. For instance, if you are in a subindex of your file and your macro doesn't reposition the path to the appropriate subindex, any READ, CONDITION, KEY, SELECT, QFORMAT, or RFORMAT command may be rendered useless.

As an example, let's say your PATH is PARTS BY TYPE:AC UNT. You ask the Query to execute a macro called FORD_CARBS that contains the following commands:

```
DOWN
DOWN
QFORMAT PARTS
CONDITION MAKE = "FORD"
CONDITION/AND DESCRIPTION = "CARB"
SELECT ALL
```

The Query begins execution of the macro's contents:

```
>> FORD_CARBS↵
> > DOWN
Path is PARTS BY TYPE:AC UNT:21110
> > DOWN
Oops! No subindex under the current path
```

The second command in the macro (DOWN) fails because the macro was designed to run from the top of the index structure. However, the Query continues with execution of the macro:

```
> > QFORMAT PARTS
The current qformat is PARTS
> > CONDITION MAKE = "FORD"
Condition is: MAKE = "FORD"
> > CONDITION/AND DESCRIPTION = "CARB"
Condition is: MAKE = "FORD"
        AND DESCRIPTION = "CARB"
> > SELECT ALL
NO records selected
```

Because the macro didn't contain accurate reposition commands, the Query was unable to select any records.

To correct this macro, you would replace the two DOWN commands with a PATH PARTS and a DOWN command.

When executing the macro, the Query would then know to first correctly position the PATH to the beginning of the PARTS subindex, and then proceed with the selection and display commands.

Therefore, as a general rule, you should include explicit index keying or PATH positioning at the beginning of any macro designed to select and/or display records. Also, you must remember to include all the QFORMAT, RFORMAT, and CONDITION statements that a self-contained macro needs.

Keep the Macro in the Query

A macro must be self-contained to run properly. It shouldn't have to go outside the Query to execute its commands. The basic premise of a macro is that it does not require user input. Therefore, it's a good idea to avoid including commands that necessitate user input.

Although you could include the CLI, QFORMAT/LINEDIT or /SPEED, and RFORMAT/LINEDIT or /SPEED commands in a macro, we don't recommend it. These types of commands require user interaction. While user interaction is possible if you run the macro as part of a interactive Query session, its impossible if you run the macro in batch mode. In batch mode, either AOS or Query, these types of commands will cause an error and abort the macro.

Nesting Macros

Although a macro must be self-contained, you can include one macro within another. In other words, a macro could call up a different macro. For example, let's say you are in the Query and call up the macro called ALL_FORD_RECORDS.QU. This macro counts all the FORD records in your database:

```
>> ALL_FORD_RECORDS↵
> > FORD_PO
```

The first command in the ALL_FORD_RECORDS.QU macro calls up another macro called FORD_PO.QU. The purpose of this second macro is to count all the Purchase Order records for FORD.

```
> > PATH PO
Path is PO
> > DOWN
Path is PO:770945
> > QFORMAT OPEN PO'S
The current qformat is OPEN PO'S
> > CONDITION MAKE="FORD"
Condition is: MAKE="FORD"
> > SELECT/COUNT
        10 records selected
---- End of macro file FORD_PO.QU
```

When the first embedded macro ends, the Query continues by calling up the FORD_PARTS.QU macro, which counts all the Parts records for FORD.

```
> >
> > FORD_PARTS
> > PATH PARTS
Path is PARTS
> > DOWN
> > QFORMAT PARTS
The current qformat is PARTS
> > CONDITION MAKE="FORD"
Condition is: MAKE="FORD"
> > SELECT/COUNT
        15 Records selected
---- END OF MACRO FILE FORD_PARTS.QU
```

Notice that this macro did not clear the condition, nor did it result in an error message on the new CONDITION command. When the qformat changed, the condition automatically cleared.

When the second embedded macro ends, the Query then continues to execute the original macro:

```
> >
> > CONDITION/CLEAR
No condition is specified
> > PATH PARTS
Path is PARTS
---- END OF MACRO FILE ALL_FORD_RECORDS.(
```

Tying up Loose Ends

Once the Query executes a macro, control returns to you. If the macro has changed the qformat, the rformat, or any other specification, you may end up wasting your efforts by retreiving the wrong information. Therefore, to make your work with the Query as easy as possible, it's a good idea to include commands at the end of the macro that will

- List or clear the current CONDITION
- List or return the current PATH
- List or change the current qformat and/or rformat.

For instance, let's say you're reading records from the PO subindex:

```
>> PATH PO:770945↓
Path is PO:770945
>> QFORMAT OPEN PO'S↓
The current qfromat is OPEN PO'S
>> READ ↓
PO NUM          : 770945
DESCRIPTION     : OIL FT
YEAR            : 72
MAKE            : MERC
MODEL           : MONARCH V8
PART NUM        : 621437
DATE            : 100578
QTY             : 22
RETAIL PRICE    : 2.98
WHOLESALE PRICE : 1.98
SUPPLIER        : ACCO
SUP ADDRESS     : SEATTLE WA
```

You tell the Query to execute the macro OIL.QU. This macro establishes a path to the PARTS subindex, tells the Query you're looking for the records that contain the word OIL in the DESCRIPTION field, and asks the Query to SELECT all matching records.

The Query executes the commands and displays the selected records.

```
>> OIL.QU↓
> > PATH PARTS
Path is PARTS
> > DOWN
Path is PARTS:056793
> > QFORMAT PARTS
The current qformat is PARTS
> > CONDITION DESCRIPTION="OIL"
Condition is:    DESCRIPTION = "OIL"
> > SELECT ALL
PART NUM        : 162934
DESCRIPTION     : OIL FT
YEAR            : --
MAKE            : DATSUN
MODEL           : Z-28 327
ON HAND         : 20
IN WAREHOUSE    : 12
REORDER         : 10
SUPPLIER        : STP
SUP ADDRESS     : INDIAN IN
RETAIL PRICE    : 1.98
WHOLESALE PRICE : 1.00

PART NUM        : 165590
DESCRIPTION     : OIL FT
YEAR            : --
MAKE            : WILLYS
MODEL           : JEEP V6
ON HAND         : 7
IN WAREHOUSE    : 2
REORDER         : 1
SUPPLIER        : LEE
SUP ADDRESS     : ST. PAULMN
RETAIL PRICE    : 2.80
WHOLESALE PRICE : 1.60

PART NUM        : 185621
DESCRIPTION     : OIL FT
YEAR            : --
MAKE            : CHEV
MODEL           : ALL
ON HAND         : 165
IN WAREHOUSE    : 100
REORDER         : 80
SUPPLIER        : GEN MOTORS
SUP ADDRESS     : DETROIT MI
RETAIL PRICE    : 2.49
WHOLESALE PRICE : 1.43

PART NUM        : 185701
DESCRIPTION     : OIL FT
YEAR            : --
MAKE            : AMC
MODEL           : JAVELIN V6
ON HAND         : 10
IN WAREHOUSE    : 5
REORDER         : 10
SUPPLIER        : FRAM
SUP ADDRESS     : CHICAGO IL
RETAIL PRICE    : 2.15
WHOLESALE PRICE : 1.25
```

And so on, until the Query finishes executing the macro and displays the following message:

```
---- End of Macro file OIL.QU
```

You then continue with your Query session, with a READ 2 command:

```
>> READ 2↵
PART NUM          : 986533
DESCRIPTION       : IG PTS
YEAR              : 66
MAKE              : RAMBLR
MODEL             : ALL
ON HAND           : 5
IN WAREHOUSE      : 5
REORDER           : 3
SUPPLIER          : AMC
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 1.42
WHOLESALE PRICE   : 1.16
```

However, the Query only reads one record because it's the only one left in the index. You can check this with a NEXT command:

```
>> NEXT↵
Alas - no more entries in this subindex
```

The PATH, CONDITION, and qformat are still as they were specified by the macro. You now have to respecify and start again.

Suggestions for Using Macros

We mentioned earlier that when a Query session becomes a repeated operation, you can build a macro of that session for later use. Here are some suggestions for building macros in your own Query facility.

You could create macros that would

1. Select and display records you often reference.
2. Specify a long or complex PATH.
3. Create a complicated CONDITION statement.
4. Give you a listing of the contents of your indexes.
5. Specify all the commands you need to perform a long or involved READ.
6. List the keys in your file that contain subindexes.
7. List the names and display the contents of all your existing qformats.
8. Contain an updated account of the commands you used during previous Query sessions.
9. Create reports.

You'll undoubtedly find many additional uses for Query macros. But most important, by building macros, you'll shorten the amount of keyboard time at your terminal; you'll thus make the Query facility a more dependable, efficient, and time-saving data processing tool.

Summary

In this chapter, we have shown how to tailor the Query facility to meet your data inquiry needs. By now, you should be able to create and edit qformats in the .QFORMS file. These qformats work effectively with the CONDITION and SELECT statements that you declare.

You should also be able to create and edit rformats in the .RFORMS file. These rformats allow you to produce a variety of reports using the data in your database.

And we've shown you how to build macros that will save important Query sessions for later use.

Chapter 4 supplies reference material about data definition in the Query's qformat file. Chapter 5 presents reference material about data definition in the Query's rformat file. In Chapter 6, we detail all the Query commands, arguments, and switches. And in Chapter 7, we'll describe how Caucus Car Parts used the Query in its everyday business operations.

End of Chapter

Chapter 4
Query Reference Section: The .QFORMS File

The INFOS system Query uses a qformat file to define the structure of INFOS file records. Although the INFOS system does not require data definition when you create INFOS records, the Query uses defined data in order to reference and display records. Breaking records into fields, via qformats, enables you to make better use of the Query.

We have tried to make the data definition process as simple and convenient as possible, while still allowing flexibility. We include a sample COBOL program that illustrates this process.

In Chapter 3, we described how you can create, edit, and delete individual qformats. In this chapter, we describe the formatting procedure and give you the reference material you'll need to build the .QFORMS file for your own application.

Because the .QFORMS file serves as an interface between the Query and the INFOS file, we suggest that someone with prior application systems experience create this file.

Building the .QFORMS File

You store the data definitions for your INFOS file as text in the <index> .QFORMS file, where <index> is the name of your working INFOS index. For example, CAUCUS.QFORMS contains the data definitions for the INFOS file CAUCUS. You can build this text file either from the CLI with a text editor or in the Query facility with either the QFORMAT/LINEDIT or QFORMAT/SPEED command, as explained in Chapter 3.

Syntax of the Query Format File

You will probably define many different qformats in the .QFORMS file. These qformats can represent unique record types or different views of the same record type. In either case, each qformat includes a list of field descriptions that contain the following information:

TITLE	A name for the field, up to 30 characters in length.
START BYTE	The byte on which the field starts.
END BYTE	The byte on which the field ends.
TYPE	The data type within the field (an optional entry).

NOTE: The system assumes that the field is in ASCII characters unless you specify otherwise.

The .QFORMS file consists of one or more qformats. A qformat consists of three types of lines: one *START_FORMAT* line, one or more *Field Descriptor* lines, and one *END_FORMAT* line.

Although you do not have to position these lines in specific columns, you must tab (< tab >) to delimit the fields, and you must use NEW LINE (↓) to delimit lines.

START_FORMAT Line

The first line of each qformat is the START_FORMAT line. This line contains two fields: the keyword START_FORMAT, followed by a tab, and the qformat name that contains up to 32 characters, including blanks, (without tabs) and followed by a NEW LINE. For example:

```
START_FORMAT<tab>THIS IS A NAME↓
```

Field Descriptor Lines

Each qformat contains from one to 30 field descriptor lines after the START_FORMAT line. The field descriptor lines contain either three or four fields, depending on whether you include the "type" field for non-ASCII formatting.

The first field is a title of up to 30 characters (but without a tab), followed by one or more tabs. The second field contains the start-byte integer, followed by a tab. The third field contains the end-byte integer followed by either a tab if you include an optional data type, or a NEW LINE if you do not include an optional data type.

In all field descriptor lines, the start_byte integer must be less than or equal to the end-byte integer. If the start-byte integer is greater, the Query will not accept the qformat, and you will receive an error message.

Both start- and end-byte fields may contain leading blanks or tabs so you can set up the integers in columns.

The fourth, optional field is the data type field containing one of the following data types:

DECIMAL TSO
DECIMAL LSO
DECIMAL LSS
DECIMAL TSS
DECIMAL
BIN INTEGER
BIN FLOAT
PACKED
ASCII

Where:

TSO is Trailing Sign Overpunch
LSO is Leading Sign Overpunch
LSS is Leading Sign Separate
TSS is Trailing Sign Separate

The data type you specify must be followed by a NEW LINE. You should be sure to spell the data type correctly and to type it in its full form. A misspelling will cause an incorrect format line.

Remember, if you do not specify an optional data type (see "Non-ASCII Formatting," below), then the field will be ASCII by default. For example:

FIRST.....FIELD < tab > 1 < tab > 15 ↵

SECOND FIELD < tab.tab > 22 < tab > 28 < tab > ASCII ↵

LAST ONE < tab....tab > 51 < tab > 56 < tab > DECIMAL LSS ↵

Even though only the second field is specifically designated both the first and second fields are ASCII data types. The last field is a DECIMAL LSS data type.

END_FORMAT Line

After you have specified all the desired field descriptor lines, you must include a last line that says END_FORMAT, followed by a NEW LINE (↵). For example:

```
END_FORMAT↵
```

Each qformat may contain up to 30 field descriptor lines. If you exceed this limit, the Query will not accept your format, and you will receive an error message:

Too many field descriptor lines in qformat

Setting Up a .QFORMS File

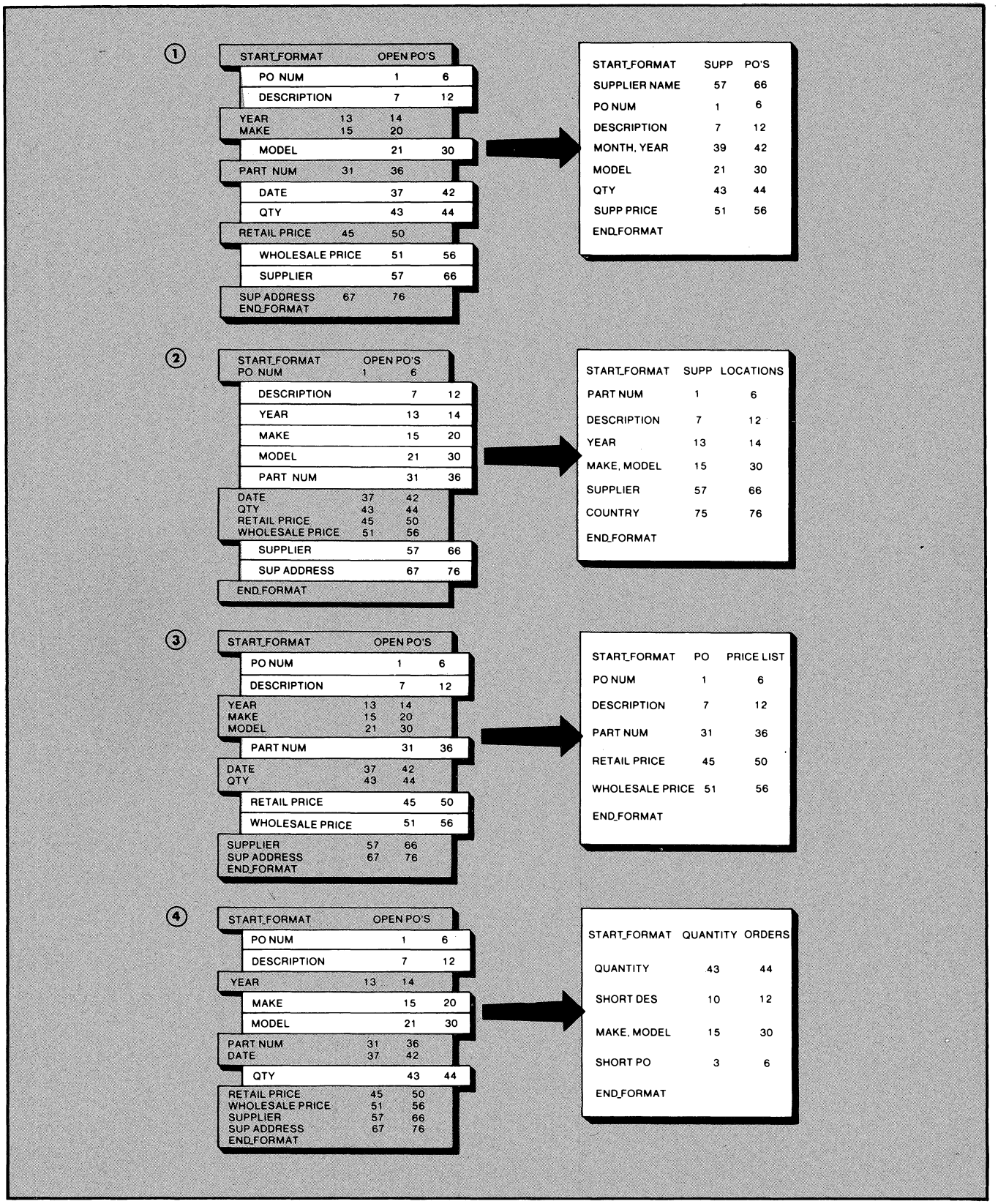
You can create different qformats for a record and display only those fields of interest in a given Query application. In the following sections we'll discuss the procedures you take to create qformats which display the fields you want. We'll also discuss non-ASCII formatting and qformat related errors.

Selection Procedures

Keep in mind that you may arrange field descriptor lines in any order. You'll find this useful when you want fields printed in a different order than they appear in the original record.

For example, Figure 4-1 illustrates some of the different qformats you can extract from a Caucus Car Parts PO record.

The Query's selection procedures allow you to make full use of possible qformats and conditions. In Chapter 3, we explained how you create qformats that will mask unnecessary parts of individual record fields. When you specify a CONDITION, the Query checks whether the matching formatted field corresponds to the one specified in the condition statement by making a normal string comparison using ASCII character values, or, in the case of non-ASCII fields, it does a numeric comparison.



SD-01343

Figure 4-1. Building Qformats

When comparing strings for equality, the Query truncates excess characters, making the two strings equal length. This exception from normal string comparison permits you to search for abbreviated parts of individual record fields. For example, in a test for equality, the string "FORD" would match any of the following:

"FORD"
"FORDXYZ"
"FORD_____"

but it would NOT match

"FOR_"
"_ORD"
"_FORD"

By carefully tailoring your qformats and condition statements to fit your records, you will greatly increase the range and versatility of your queries.

Non-ASCII Formatting

It is easiest to use qformats when all the data records are ASCII values. Because ASCII is the default display mode, you only name the different byte ranges that you need to access.

Many files, however, especially those created and maintained by COBOL programs, contain non-ASCII data. If you were to display these records directly on a display terminal without changing the data type, they would appear visually incomprehensible. By using the appropriate field description types, the INFOS Query converts the values into readable form (numbers).

To use this Query feature for converting non-ASCII data, you will have to know how many bytes are in the internal representation of each data type. Then, you can correctly position the start- and end-bytes of the respective fields.

Table 4-1 shows the possible data type descriptions that the Query can use with possible COBOL picture clauses for each type, and the number of bytes required in the field.

The following COBOL program, CISTYPES, creates an INFOS database, CISTYPES.DB, that contains records filled with commercial data types.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CISTYPES.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT OFILE
          ASSIGN INDEX TO 'CISTYPES.IX'
          ASSIGN DATA TO 'CISTYPES.DB'
          ORGANIZATION IS INDEXED
          ACCESS MODE IS DYNAMIC
          RECORD KEY IS KEY1.

DATA DIVISION.
FILE SECTION.
FD      OFILE.
01      REC-1.
02 KEY1 PIC X(4).
02 F-1 PIC S9(5).
02 F-2 PIC S9(6) SIGN IS LEADING.
02 F-3 PIC S9(7) SIGN IS LEADING SEPARATE.
02 F-4 PIC S9(8) SIGN IS TRAILING.
02 F-5 PIC S9(5) SIGN IS TRAILING SEPARATE.
02 F-6 PIC 9(5) USAGE COMP.
02 F-7 PIC 9(12) USAGE COMP.
02 F-8 USAGE COMP-1.
02 F-9 USAGE COMP-2.
02 F-10 PIC 9(14) USAGE COMP-3.
WORKING-STORAGE SECTION.

PROCEDURE DIVISION.

BEGIN.
        OPEN  OUTPUT OFILE.
        MOVE 0 TO KEY1.
        MOVE 1 TO F-1.
        MOVE 10 TO F-2
        MOVE 20 TO F-3.
        MOVE 30 TO F-4.
        MOVE 40 TO F-5.
        MOVE 50 TO F-6.
        MOVE 60 TO F-7.
        MOVE 70 TO F-8.
        MOVE 80 TO F-9.
        MOVE 90 TO F-10.
        PERFORM LOOP 10 TIMES.

LOOP.
        COMPUTE KEY1 = KEY1 + 1.
        COMPUTE F-1 = F-1 + 1.
        COMPUTE F-2 = F-2 + 1.
        COMPUTE F-3 = F-3 + 1.
        COMPUTE F-4 = F-4 + 1.
        COMPUTE F-5 = F-5 + 1.
        COMPUTE F-6 = F-6 + 1.
        COMPUTE F-7 = F-7 + 1.
        COMPUTE F-8 = F-8 + 1.
        COMPUTE F-9 = F-9 + 1.
        COMPUTE F-10 = F-10 + 1.
        WRITEREC-1.

ALLDONE.
        CLOSE  OFILE.
        STOP  RUN.
```

Table 4-1. Type Descriptions

Datatype	COBOL Equivalents	Bytes Required
DECIMAL TSO	PIC S9(n) PIC S9(n)SIGN IS TRAILING	n bytes n bytes
DECIMAL LSO	PIC S9(n)SIGN IS LEADING	n bytes
DECIMAL TSS	PIC S9(n)SIGN IS TRAILING SEPARATE	n + 1 bytes
DECIMAL LSS	PIC S9(n)SIGN IS LEADING SEPARATE	n + 1 bytes
DECIMAL	PIC 9(n)	n bytes
BIN INTEGER	PIC 9(n) USAGE IS COMP	1 byte for n=1,2 2 bytes for n=3,4 3 bytes for n=5,6 4 bytes for n=7,8,9 5 bytes for n=10,11 6 bytes for n=12,13,14 7 bytes for n=15,16
BIN FLOAT	USAGE COMP-1 USAGE COMP-2	8 bytes 8 bytes
PACKED	PIC 9(n) USAGE COMP-3	(n+1)/2 rounded up bytes

As you can see, this program contains some of the picture clauses listed in Table 4-1. By using the conversion information in Table 4-1 to determine the data type byte requirements, you can create the following qformat file that the Query will use to make the records in the database readable.

```
START_FORMAT    XXX
KEY  1    4    ASCII
F1   5    9    DECIMAL TSO
F2   10   15   DECIMAL LSO
F3   16   23   DECIMAL LSS
F4   24   31   DECIMAL TSO
F5   32   37   DECIMAL TSS
F6   38   40   BIN INTEGER
F7   41   46   BIN INTEGER
F8   47   54   BIN FLOAT
F9   55   62   BIN FLOAT
F10  63   70   PACKED
END_FORMAT
```

This qformat file instructs the Query to convert the non-ASCII data types into readable ASCII form. Now, when you query the CISTYPES.IX index, you are able to read the non-ASCII records. For example:

```
INFOS Query / Report Writer ...At Your Service
>> QFORMAT/NAMES↵
The following qformats have been defined on this file :
        XXX
>> QFOR X↵
The current qformat is XXX
>> QFOR/DIS↵
KEY         1    4    ASCII
F1          5    9    DECIMAL TSO
F2          10   15   DECIMAL LSO
F3          16   23   DECIMAL LSS
F4          24   31   DECIMAL TSO
F5          32   37   DECIMAL TSS
F6          38   40   BIN INTEGER
F7          41   46   BIN INTEGER
F8          47   54   BIN FLOAT
F9          55   62   BIN FLOAT
F10         63   70   PACKED DEC
>> PA↵
Path is above the index
>> DOWN↵
Path is 0001
>> READ↵

KEY          : 0001
F1           : +000000000000002
F2           : +000000000000011
F3           : +000000000000021
F4           : +000000000000031
F5           : +000000000000041
F6           : +000000000000051
F7           : +000000000000061
F8           : 7.099999E+01
F9           : 8.100000E+01
F10          : +000000000000091
```

Qformat Error Messages

Each time you start a Query session, and after every QFORMAT/SPEED or QFORMAT/LINEDIT session, the Query will read through the .QFORMS file and verify that all the lines are syntactically correct. If the Query finds no errors, it will return a prompt (>>) and you can continue querying the file. However, if there are errors, then the Query will respond with the appropriate error messages. You will find a complete listing of qformat error messages in Appendix B.

Runtime Format Errors

There are two cases where the Query detects errors in the formatted records during a READ or a SELECT command. In both cases, the Query prints an error message instead of the data value.

Record too short for field

This message means the END BYTE specified in the qformat for this field is greater than the length of the record read from the INFOS file. This error could occur if you have multiple record types in a given file and the type of the record read does not correspond with the current qformat.

Commercial Fault n

Where *n* is an integer from 1 to 9.

This message appears when the Query attempts to read a commercial data type specified in the current qformat, and finds the bytes indicated are not of the correct type. This error would occur if you miscalculated the byte lengths or chose the wrong data type.

Size and Scope of the .QFORMS File

Your .QFORMS file may contain up to ten separate qformats. If you need more space for your qformats (without deleting any you have already defined), you can create either another .QFORMS file in a different directory or another file with a different name in the same directory using the CLI. Note, however, that only the < index > .QFORMS file that corresponds to your index may be current at any given time.

For example, if the CAUCUS.QFORMS file already contains ten qformats, you could

1. Create another file, CAUCUS.QFORMS_II, that contains any additional qformats. When you need to access this file, you simply switch the existing filenames (with the CLI RENAME command) before calling the second file from the Query.

2. Create another CAUCUS.QFORMS file in a different directory using the CLI. When you need to access the alternate file, you simply swap the files between the two directories (using the CLI MOVE command) before specifying them in the Query.

3. Create another file, possibly called CAUCUS.QF_I that contains any additional qformats. Before entering the Query, create a link to this file, using the CLI CREATE/LINK command. Anytime you want to use another file that contains different qformats, you simply delete the link and create a new one.

You also may want to periodically weed out your .QFORMS file by deleting any qformats you no longer need. Ideally, you should try to establish an assortment of qformats that will serve as a foundation for your Query activities.

Summary

In this chapter, we have described the process by which you create, organize, and use the Query qformat file. We have also described how you build qformats so the Query can display non-ASCII files. And finally, we have explained how the Query will check your work, and how you can correct any possible errors.

End of Chapter

Chapter 5
Report-Writer Reference Section: The .RFORMS file

The Report Writer takes INFOS database records and rearranges them into report forms in an output file, along with whatever headings, page breaks, and computations you want the file to contain.

An rformat defines the eventual report the Report Writer produces. In Chapter 3, we described how you can create, edit, and delete individual rformats. In this chapter, we describe the creation of rformats in more detail, and give you the reference material you'll need to build the .RFORMS file for your application.

As you read this chapter, consider how the Report Writer uses the rformat. When invoked, the Report Writer reads through the rformat, making note of each different descriptor line and the action each requires. It then reads one INFOS record at a time and performs the actions the rformat has requested. We introduced this concept in Chapter 3 and will reiterate it as we explain the different format descriptor lines that make up an rformat.

The data definition needs of the Report Writer are the same as those of the Query. In other words, the Report Writer uses the same .QFORMS file the Query does. You should understand qformats, especially those designed for your specific INFOS file, before reading this chapter. If you are not familiar with the qformats designed for your system, or with the .QFORMS file and how to build it, see Chapter 4.

Building the .RFORMS File

You create rformats in an AOS text file called <index>.RFORMS, where < index > is the name of your current INFOS index. You can build this text file from the CLI with a text editor or in the Query facility with either the RFORMAT/LINEDIT or RFORMAT/SPEED commands.

Rformat Definition

An rformat consists of a number of different *format descriptor lines*. Each line has a limited number of fields, separated by tabs. As we describe each line, you'll see how flexible rformat definition is. The syntax you use to define rformats is loosely based on the qformats the Query uses. You must follow certain simple rules when creating rformats; however, the resulting reports can be as complex or simple as you want.

Rformats must begin with a START_REPORT line and end with an END_REPORT line. As we shall see, between these first and last lines, the other types of format descriptor lines may appear. Blank lines may appear anywhere within the rformat. In fact, you should feel free to use blank lines often as a means of grouping statements. This will make the rformat easier to read.

As we explain the individual format descriptor lines, keep in mind that these lines tell the Report Writer what actions to take, the types of information that need to be displayed, and what the size and shape of the report should be.

The command line description and terminology used in defining rformats follows:

UPPERCASE LETTERS	Keywords
lowercase letters	Variables
<tab>	Tab:delimiter between arguments
⤸	NEW LINE: end-of-line delimiter
data_item	A data field described in the qformat file
def_item	A data field described by a DEFINE line

We will introduce some basic rules about creating rformats here, and then explain them in further detail later on.

Although there are various Report Writer format descriptor lines that you must include in the rformats, the order of these lines is highly flexible. You can group these lines so they are easy for you you to read and understand. Suit yourself. Just follow the basic rules, and the Report Writer will check for syntax errors.

- The rformat always begins with a START_REPORT command and ends with an END_REPORT command.
- The QFORMAT command line must precede any command line that makes reference to a data_item.
- You must define a field in a DEFINE line before using it any in another line.
- You can define a def_item only once per rformat.
- Any item (def or data) can only have one PICTURE per rformat.
- COL/LIN and LIN/PAGE command lines each appear only once in an rformat.
- A DETAIL command line for an item must precede any BREAK command line calling for underlining that item.

START_REPORT Line

The first line of each rformat is the START_REPORT line. It signals the beginning of an rformat by naming it.

This line contains two fields: the keyword START_REPORT, followed by a tab, and the name of the report which contains up to 32 characters including blanks with no tabs, followed by a NEW LINE. For example;

START_REPORT <tab> report_name↵

Keeping the rformat name unique is a good idea. Let's say you have one rformat called STOCK VALUE and one called STOCK INVENTORY. If you try to refer to either rformat by STOCK, the Report Writer will interpret it as the first one defined in the .RFORMS file. This could cause some confusion if you try to use the RWCHECK using the /R switch (explained later in this chapter). Therefore, either use one word names for rformats or make sure the first word of the name is unique.

Comment Line

If you want to make notes when building a rformat, use the comment line. The comment line gives no instructions to the Report Writer; it's for documentation purposes only.

The comment line consists of an exclamation point (!) in the first column followed by your message. You can include any number of comment lines in a rformat. For instance, the person at Caucus Car Parts who built the STOCK VALUE rformat included the following comments:

```
!
! This produces a report with the total value of the current
! warehouse stock
!
```

These comments will not show up on the report produced by this rformat. However, anyone who looks at the .RFORMS file will be able to read them.

QFORMAT Line

The QFORMAT command line specifies which qformat you want the Report Writer to use when it accesses the records from the database. Each qformat defines different data items. The QFORMAT line allows you to refer to the various data items by the names used in one of these qformats. You must have defined the qformat before you specify it in the QFORMAT command line.

This line consists of two fields: the keyword QFORMAT, followed by a tab, and the qformat name, followed by a NEW LINE.

For example:

QFORMAT <tab> qform_name↵

The rformat STOCK VALUE specifies the qformat PARTS.

Lines Per Page

You can specify how many lines per page you wish the report to contain. The default is 60 lines per page; the minimum allowed is 30. Keep in mind your system's line printers may differ in terms of top and bottom margins. For instance, if a line printer is automatically set to leave a three line margin at top of the page, given a 66 line page, only 63 lines are available for printing. Check with your system manager when and if you need to adjust these margins, or see *The AOS Command Line Interpreter User's Manual* (093-000122) for more details about the Forms Control Utility.

The line consists of two fields: the keyword LIN/PG, followed by a tab, and the number you wish to specify, followed by a NEW LINE. The LIN/PG line takes the following form:

LIN/PG <tab> n↵

where n is the number of lines per page.

Columns Per Line

You can also specify the width of the printed report page, in number of characters. The default is 80 characters.

This line also has two fields: the keyword COL/LIN, followed by a tab, and the number you wish to specify, followed by a NEW LINE. The COL/LIN line takes the following form:

COL/LIN <tab> n↵

If you plan to send all the output for a given report to the terminal you must specify 80 or less COL/LIN. Otherwise, each line of output will wrap around on the terminal screen, making it difficult to read. If you plan to print the report, check to see what width paper is used on the lineprinter and set the COL/LIN appropriately. In addition, if you want to use the /SCREEN switch with the READ and/or SELECT commands, don't define more than 80 characters per line.

HEADER Lines

You use HEADER lines to define report and column headers. HEADER lines tell the Report Writer to display header information at the top of each page of the report. In other words, if the printed report runs more than one page, the header information appears on each page.

You can define up to ten lines of header information in an rformat. You don't have to define them in any particular order, or all together. The Report Writer will check that items on the same HEADER line do not overlap.

The HEADER line consists of four fields: the keyword HEADER, the line number on which the header information is to appear, the location on the line where the header information is to appear, and what header information is to print there. A tab follows the first three fields, a NEW LINE follows the last field. A HEADER line would look like the following:

HEADER <tab> line # <tab> loc <tab> item↵

Where:

loc is	C centered n start at byte n -n start n bytes from the end
item is	"literal" 'literal' data_item DATE (today's date) PAGE (the current page number)

Note: The PAGE field is ten characters long. The DATE field is eight characters long.

You can have more than one HEADER line for the same line number; the Report Writer combines these specifications to determine the final line. For instance, if you want to define five column heads on the same line, your HEADER lines might look like the following:

```
HEADER 6    5    "Description"
HEADER 6    20   "Part Number"
HEADER 6    35   "Stock"
HEADER 6    45   "Cost"
HEADER 6    60   "Dollar Value"
```

Although you have used five HEADER lines, you have told the Report Writer to actually display only one line of HEADER information.

These five HEADER lines do not need to appear in this order or even contiguously. In fact, in the rformat called STOCK VALUE, these lines are scattered throughout the file.

If you have more than one line of HEADER information in your report, you do not have to specify all the line numbers. If you skip a line number, the Report Writer will interpret it as a blank line in your printed report. For instance, if you have three lines of HEADER information defined to appear on lines 1, 3, and 5, the Report Writer will interpret lines 2 and 4 as blanks. After all the lines of HEADER information have been displayed, the Report Writer skips one more line before displaying the DETAIL information. You do not have to specify a blank line between the HEADER information and the DETAIL information; the Report Writer automatically does it for you.

If you specify a data_item as part of your HEADER information, the value of that specified item will be the same as its value on the first DETAIL line for that page. In other words, the Report Writer finds that item's value from the same record it uses for the first line of DETAIL information in the report.

DEFINE Lines

You use DEFINE lines to define temporary variables which are considered part of the record for the duration of the Report Writer process. You're more or less asking the Report Writer to temporarily extend the record to include new items. The Report Writer performs calculations to derive these temporary items.

This line consists of three fields: the keyword DEFINE, followed by a tab, a defined item, a tab, a simple arithmetic expression, and a NEW LINE. The DEFINE line takes the following form:

DEFINE <tab> def_item <tab> expression↵

An expression involves exactly one operator (+, -, /, *, %) and any two of the following: data_items, def_items, and numeric constants.

The temporary variable defined in the DEFINE line is called a def_item. The results of the calculations resulting from a DEFINE line are not retained from record to record; they are calculated anew for every record. You may define up to ten temporary variables.

For instance, if an input record has the fields, PART_NBR, QUANTITY, and PRICE, you might use the following DEFINE lines:

```
DEFINE COST     PRICE * QUANTITY
DEFINE PROFIT   2 % COST
```

In this case, you can ask the Report Writer to display any of the six items, including COST and PROFIT, at some point in the report.

DETAIL Lines

You use DETAIL lines to specify which items from each record you want printed in the report. A DETAIL line tells the Report Writer what to include as DETAIL information. You may define up to 5 lines of DETAIL information per report. As with the HEADER line, you may specify more than 5 DETAIL lines lines in order to create 5 lines of DETAIL information. The Report Writer checks for overlapping fields on every DETAIL line.

The DETAIL line consists of four fields: the keyword DETAIL, the number of the DETAIL line (1 through 5), where on the page the DETAIL information will appear, and the items you are specifying. The first three fields are followed by a tab; the item itself is followed by a NEW LINE. The DETAIL line takes the following form:

DETAIL <tab> line # <tab> loc <tab> item↵

Where:

line # is number of the DETAIL line 1 through 5

loc is	C centered n start at byte n -n start n bytes from the end
item is	"literal" 'literal' data_item def_item

In the rformat called STOCK VALUE, five different items from the same record were needed for the report.

A different DETAIL line is used for each item, even though all five items will appear on the same line in the report. The same number, 1, appears in each DETAIL line, indicating it is DETAIL line 1. This rformat could have defined up to four more lines of DETAIL information.

PICTURE Lines

You use the PICTURE line to specify the way you want a particular item to appear in the printed report. If you do not define a PICTURE line, the Report Writer outputs any numeric data in a 14 character field with 10 signficant digits and two digits to the right of the decimal. By numeric data we mean all data_items described as numbers in the qformat (not ASCII), and all def_items.

The PICTURE line consists of three fields: the keyword PICTURE, followed by a tab, an item name followed by a tab, and the picture clause, followed by NEW LINE. The PICTURE line takes the following form:

PICTURE <tab> item <tab> pic↵

Where:

item is	a data_item a def_item
pic is	a picture clause

PICTURE allows you to specify a field format using a subset of the PL/1 and COBOL picture facilites. The Report Writer recognizes seven symbols, as explained in Table 5-1.

Table 5-1. Picture Characters

Picture Character	Symbol Definition	Usage
9	Numeric digit	Indicates a numeric digit, 0 through 9.
$	Dollar Sign	Indicates a currency symbol or a numeric digit.
V	Floating decimal point	Indicates decimal point location.
S	Signed number	Indicates a sign or a digit
/	Slash	Indicates a slash character.
.	Period	Indicates a period character.
,	Comma	Indicates a comma character.

A $, if present, must always precede 9s and Vs. Only one V is legal per picture. An S may only be at either end of a picture, while floating Ss are always at the beginning. Floating signs (S) imply no legal $s and floating dollar signs ($) imply one S at most. Commas (,), periods (.), and slashes (/) are simply cosmetic characters and may go anywhere in the picture.

For example, here are five different picture clauses for the signed number -43.20:

SSSS9.V99	would result in -43.20
S999.V99	would result in - 43.20
99V.99S	would result in 43.20-
S99V99	would result in -4320
S9999	would result in - 43

SORT Lines

You use SORT lines to specify the order of the records that will appear in the report. You specify the data item you want used as a sort key, and whether or not you want the records sorted in descending order. The default is sorting in ascending order. If you don't want to take the default, you must follow the data item by the keyword DESCENDING or a unique abbreviation.

The SORT line consists of two mandatory fields, followed by the tab, and the optional DESCENDING field, followed by a NEW LINE. The mandatory fields are the keyword SORT and the data item. The SORT line takes the following form:

SORT <tab> data_item <tab> *[DESCENDING]* ↓

If you use more than one SORT line in an rformat, the sort keys are taken in the same order as the lines appear in the rformat.

You should be aware that the SORT lines invoke the AOS SORT utility which can be very time consuming. Therefore, you should not use SORT lines when they are not necessary. For instance, you would not need a SORT line if the records will be drawn from an INFOS subindex in the desired order.

BREAK Lines

A BREAK line tells the Report Writer to suspend all other actions long enough to take a specified action. There are several actions which you may optionally specify:

- print a line of summary information.
- suppress repeated fields in DETAIL lines (NO_REP).
- skip to a new page, or to a new line (POST_BREAK_SPACE, PRE_BREAK_SPACE, PAGE_EJECT).
- underline specific columns (UNDERLINE).

The Report Writer performs the BREAK action when it encounters a change in the value of a specified data item. Any summary information printed by a BREAK action can take only one physical line on the report. However, you can specify BREAK actions on up to five different data items. In other words, you can tell the Report Writer to take a BREAK action when up to five data values change.

A BREAK line can take one of two different syntactic forms: in one form it tells the Report Writer to print summary information at the time of the BREAK action; the other form tells the Report Writer about optional printing or spacing features.

In either case, the first two fields consist of the keyword BREAK, followed by a tab, and the name of the data item whose changing value will cause the BREAK action to occur, also followed by a tab.

The BREAK line takes the following form:

BREAK <tab> data_item <tab> loc <tab> item↵

or

BREAK <tab> data_item <tab> print_op↵

Where:

loc is	C centered n start at byte n -n start n bytes from the end
item is	"literal" 'literal' data_item COUNT TOTAL (data_item) AVG (data_item) MIN (data_item) MAX (data_item) def_item TOTAL (def_item) AVG (def_item) MIN (def_item) MAX (def_item)
print_op is	NO_REP POST_BREAK_SPACE PRE_BREAK_SPACE PAGE_EJECT UNDERLINE (data_item) UNDERLINE (def_item)

The keyword PRE_BREAK_SPACE tells the Report Writer that you want a blank line *before* any summary information. The keyword POST_BREAK_SPACE directs the Report Writer to put a blank line *after* printing summary information. PAGE_EJECT tells the Report Writer to eject a page after all BREAK actions have been taken. And, UNDERLINE tells the Report Writer to underline the specified item.

Note that there is not a tab between the keyword and its argument (for instance UNDERLINE (data_item)).

The data_item specified in a BREAK line does not have to appear in the DETAIL line or in the SORT line. Note, however, to make sense, the records should be sorted somehow on this key. In other words, if the records are not naturally sorted by the specifed data_item or sorted by a SORT line, any computed subtotals would be meaningless.

If multiple BREAK actions occur after the same DETAIL line, the lines of BREAK information will be printed in the order in which they appear in the rformat. All BREAK actions will be triggered after the last DETAIL and before any TOTAL lines.

In addition to printing data_items, def_items, and constants, however, the Report Writer also allows you to print summary items (total, average, minimum, maximum, and count) of data_items and def_items as BREAK information. The summary item calculations are done by the Report Writer; you can not maintain these calculations. The Report Writer will check that no fields overlap on any line of BREAK information.

If the BREAK line describes optional printing or spacing features, a keyword will generally suffice to tell the Report Writer what is desired. However, UNDERLINE requires a reference to a DETAIL line field as well, so the Report Writer knows what to underline.

TOTAL Lines

TOTAL lines tell the Report Writer to print summary information at the end of the report. TOTAL information can occupy up to 10 lines on the report and can include calculations performed by the Report Writer.

TOTAL lines consist of four fields: the keyword TOTAL, the TOTAL line number, where on the line the information will appear, and the information you want in the TOTAL line. The first three fields are followed by a tab; the last field is followed by a NEW LINE.

The TOTAL line takes the following form:

TOTAL <tab> line # <tab> loc <tab> item↵

Where:

line # is the TOTAL line number

loc is	C centered n start at byte n -n start n bytes from the end
item is	"literal" 'literal' data_item COUNT TOTAL (data_item) AVG (data_item) MIN (data_item) MAX (data_item) def_item TOTAL (def_item) AVG (def_item) MIN (def_item) MAX (def_item)

TOTAL information can include data_items, def_items, summary information for these items, and constants. The Report Writer does any necessary calculations required by summary items and checks that data fields don't overlap on the report's TOTAL information lines.

Although TOTAL information appears on the report only once, at the end of the report, you can use multiple TOTAL lines if the TOTAL information includes multiple fields. TOTAL information can occupy up to 10 lines on the report.

END_REPORT Line

The last line of a rformat is always an END_REPORT line. It signals the end of the rformat.

This line consists of one field, the keyword, END_REPORT, followed by a NEW LINE.

For example:

```
END_REPORT↓
```

Designing Reports

The various format descriptor lines we've just described make up an rformat. As you've noticed, some of these lines appear once per rformat, others may appear many times, and some you may not need at all. In fact, you can define an rformat with as little as a START_REPORT line, an END_REPORT line, and one DETAIL line -- or just one BREAK line or just one TOTAL line. As long as you have the starting line and the ending line, and at least one of these other lines, you'll have a legal rformat that will produce a bona fide report.

What you do with these various format descriptor lines and how you use them to define your rformat depends on the report you want to design. We can't dictate how your reports should look or what they should include. But, now you know how to design them.

As we've said before, you use a text editor to define an rformat. Once you've entered the various format descriptor lines, making sure to follow the simple rules associated with each, you can verify the rformat's syntax using RWCHECK, the Stand-Alone Compiler.

Before using your newly designed rformat to produce a report, it's a good idea to produce a sample page. In addition to verifying the syntax of any rformat you've created, RWCHECK can also provide a sample report. You'll be able to see what the report will look like; if you're not satisfied with the results, you simply edit the rformat. You can repeat these steps as many times as necessary to produce a satisfactory report format.

When you have an rformat that produces the desired report, you're ready to use it in conjunction with Query.

You can now activate the rformat during a Query session, and by using the /RFORMAT switch with the READ and SELECT commands, use the data read or selected to produce a report.

Size and Scope of the .RFORMS File

Your .RFORMS file may contain up to ten different rformats. If you try to define more than ten rformats in one .RFORMS file, the Report Writer won't recognize them. If you need more space for your rformats, (without deleting any you've already defined), you can create either another .RFORMS file in a different directory or another file with with a different name in the same directory, using the CLI. Only the < index > .RFORMS file that corresponds to your index may be current at any given time.

For example, if the CAUCUS.RFORMS file already contains ten rformats, you could

1. Create another file, CAUCUS.RFORMS_II, that contains additional rfromats. When you need to access this file, you simply switch the existing filenames (with the CLI RENAME command) before calling the second file from the Query.

2. Create another CAUCUS.RFORMS file in a different directory using the CLI. When you need to access the alternate file, you simply swap the files between the two directories (using the CLI MOVE command) before specifying them in the Query.

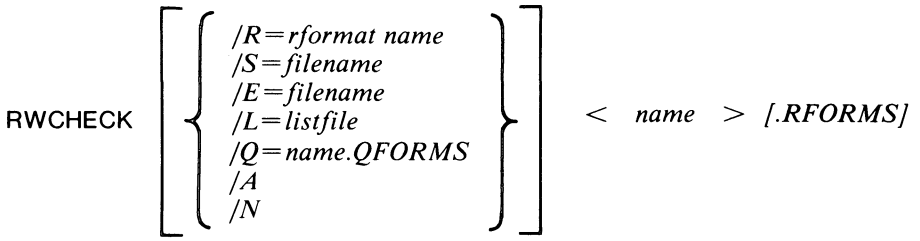
3. Create another file, possibly called CAUCUS.RF_I, that contains any additional rformats. Before entering the Query, create a link to this file, using the CLI CREATE/LINK command. Anytime you want to use another file that contains different qformats, you simply delete the link and create a new one.

You may also want to periodically weed out your .RFORMS file by deleting any rformats you no longer need. However, you should not feel limited to ten rformats; if you need more, simply create additional .RFORMS files.

RWCHECK - The Stand-Alone Compiler

RWCHECK is an interactive utility that verifies the syntax of an rformat. It also provides a sample page of your report. INFOS need not be running in order to use RWCHECK. You don't even need an INFOS file to use it. However, you do need a file containing defined qformats since you specify a qformat in your rformat definition.

You can use RWCHECK in batch mode, as well as interactively, depending on the switch used when invoking it. You invoke RWCHECK from the CLI, using the following syntax:



Where:

< name > .RFORMS Is the name of the file containing defined rformats.

/R=rformat name Allows you to specify the particular *rformat name* within the .RFORMS file you want verified.

/S=filename Allows you to designate the *filename* to which you want a sample page sent.

/E=filename Sends any errors and the lines on which these errors occur to *filename.*

/L=listfile Sends the entire rformat and any errors to a named *listfile.*

/Q=name.QFORMS Allows you to specify a .QFORMS file that is not < name > .QFORMS.

/A Specifies that all rformats in the named .RFORMS file are to be validated.

/N Specifies no interaction. No questions will appear on the screen during the RWCHECK session.

Keep in mind the following rules when you use the switches:

- You can not use a /A switch with a /R switch.
- If you use the /R switch and specify the name of an rformat, there cannot be any spaces in the rformat name. If the rformat name consists of more than one word, use only the first word of the name.
- The /N switch also requires that you use a /R or a /A switch and that you specify the .RFORMS file as an argument.
- If you do not use the /L switch, all errors are sent to @ OUTPUT.
- The /S switch tells the RWCHECK to create a sample page and send it to the named file. If you don't use this switch, you must indicate whether you want a sample page and where you want it sent during the RWCHECK session.

When you invoke the RWCHECK, an interactive dialog begins (except when you use the /N switch). During this dialog, RWCHECK asks various questions and displays the default answer in brackets ([ ]).

The only argument used with the RWCHECK command is the .RFORMS filename. If you do not include the argument, RWCHECK asks you for the name of a file. For example:

```
) X RWCHECK↓
RFORMS Filename:
```

Type the .RFORMS filename and the RWCHECK session continues:

```
RFORMS Filename: CAUCUS↓
Would you like to see a list of report format names? [N] Y↓
STOCK VALUE
OLDIES
Report Format Name: STOCK VALUE↓
```

If there are any errors, a message appears telling you on which line the error exists. For example, let's say you defined the following rformat, called BOGUS.

```
START_REPORT  BOGUS
BREAK         PRE_BREAK_SPACE
END_REPORT
```

On first glance it looks right. You've got the starting and ending lines and at least one BREAK command line. However, when you verify it with RWCHECK, the information at the bottom of this page appears on your terminal.

RWCHECK indicates an error in the BREAK line. In addition, since this is the only format descriptor line in the rformat (other than the starting and ending lines) and it's erroneous, it doesn't register as a legal format descriptor line. Therefore, you receive the second error message.

If you've defined an error-free rformat, RWCHECK responds with

No errors detected. Want a sample page? [Y]

(Note that you'll get this question only if you omitted the /S switch when invoking the RWCHECK.)

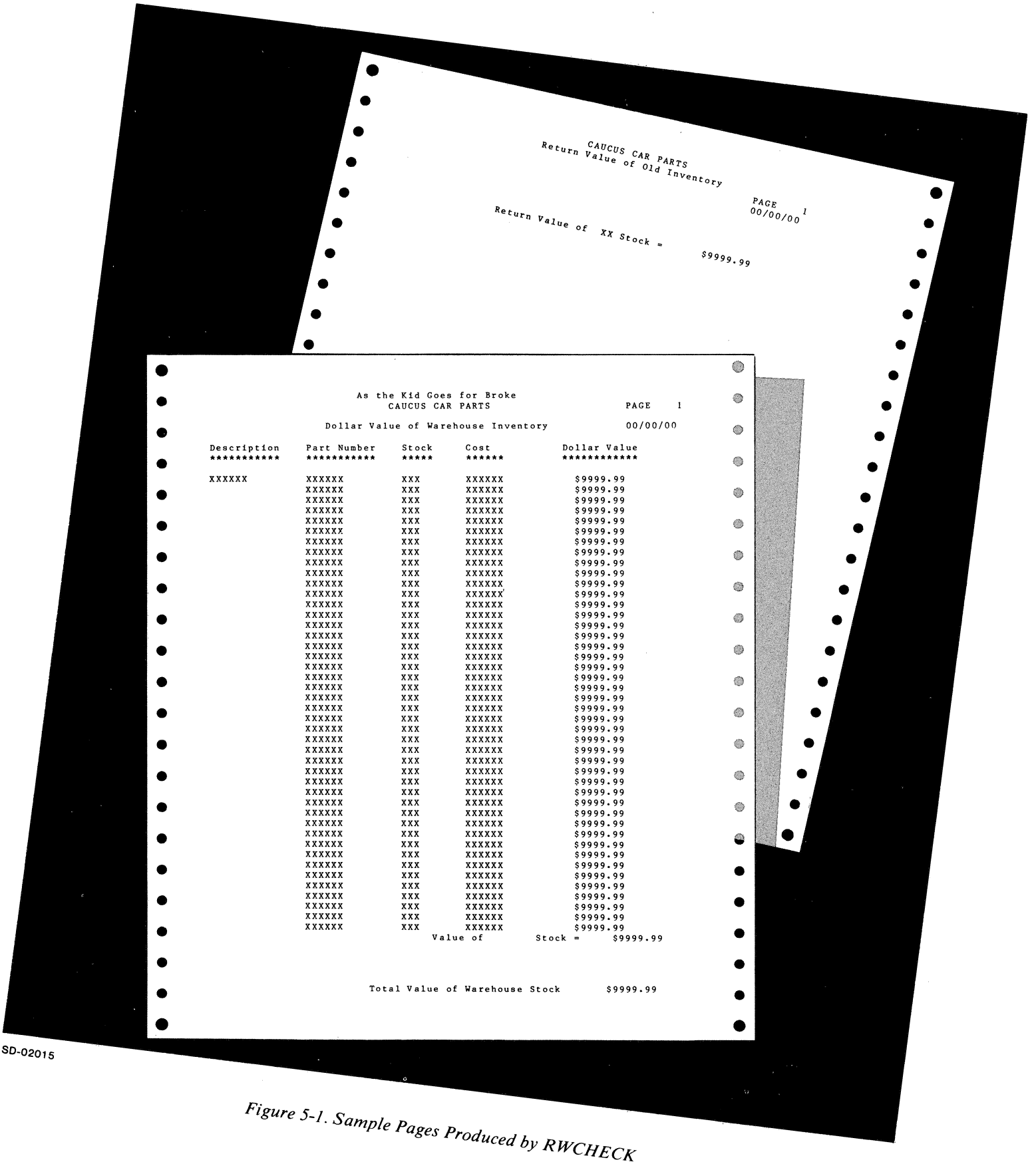
If you answer Y, the RWCHECK then asks

Name a file ...any file: [@ OUTPUT]

You can then indicate the name of a text file to which you want the sample page sent, or you can take the default and let the RWCHECK send the sample page to @ OUTPUT.

Figure 5-1 shows sample pages created by RWCHECK for the rformats STOCK VALUE and OLDIES.

```
START_REPORT BOGUS
BREAK   PRE_BREAK_SPACE
*** Wrong number of arguments for this command:  BREAK LINE:  2
END_REPORT
*** There MUST be a detail, total, or break line somewhere.
*** TOTAL ERRORS:  2
```



SD-02015

Figure 5-1. Sample Pages Produced by RWCHECK

Whether you want a sample page or not, RWCHECK next asks if you want to validate another rformat:

Do you wish to validate another report format? [N]

If you do, RWCHECK repeats the entire dialog. If you don't, the session ends, and you'll be back in the CLI.

Using Rformats

Now that you've defined the rformat that will produce the report you want, you can use it during a Query session. As explained in Chapter 2, use the /RFORMAT *switch* with the READ and SELECT commands. Use the RFORMAT *command* and its switches to activate or deactivate an rformat, to find out which rformat is currently active, to find out what rformats exist in the .RFORMS file, and to call up a text editor to create or modify an rformat. When you want to use an rformat to display database records in a report, you must activate one first (using the RFORMAT command), and then use the /RFORMAT switch with either the READ or SELECT command.

For example, you activate the STOCK VALUE rformat using the RFORMAT command:

```
>> RF STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
```

You position your path to the correct point in the index structure:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
```

Then, using the READ command, put ALL the database records in a STOCK VALUE report:

```
>> READ/TRAVERSE/RF ALL↵
```

The resulting report is shown in Figure 5-2.

Compare the report shown in Figure 5-2 with the report shown in Figure 5-3. The report in Figure 5-3 was created with the following same series of commands, except that a SELECT was used instead of a READ. Notice the system response after the first SELECT command. A CONDITION had to be set first.

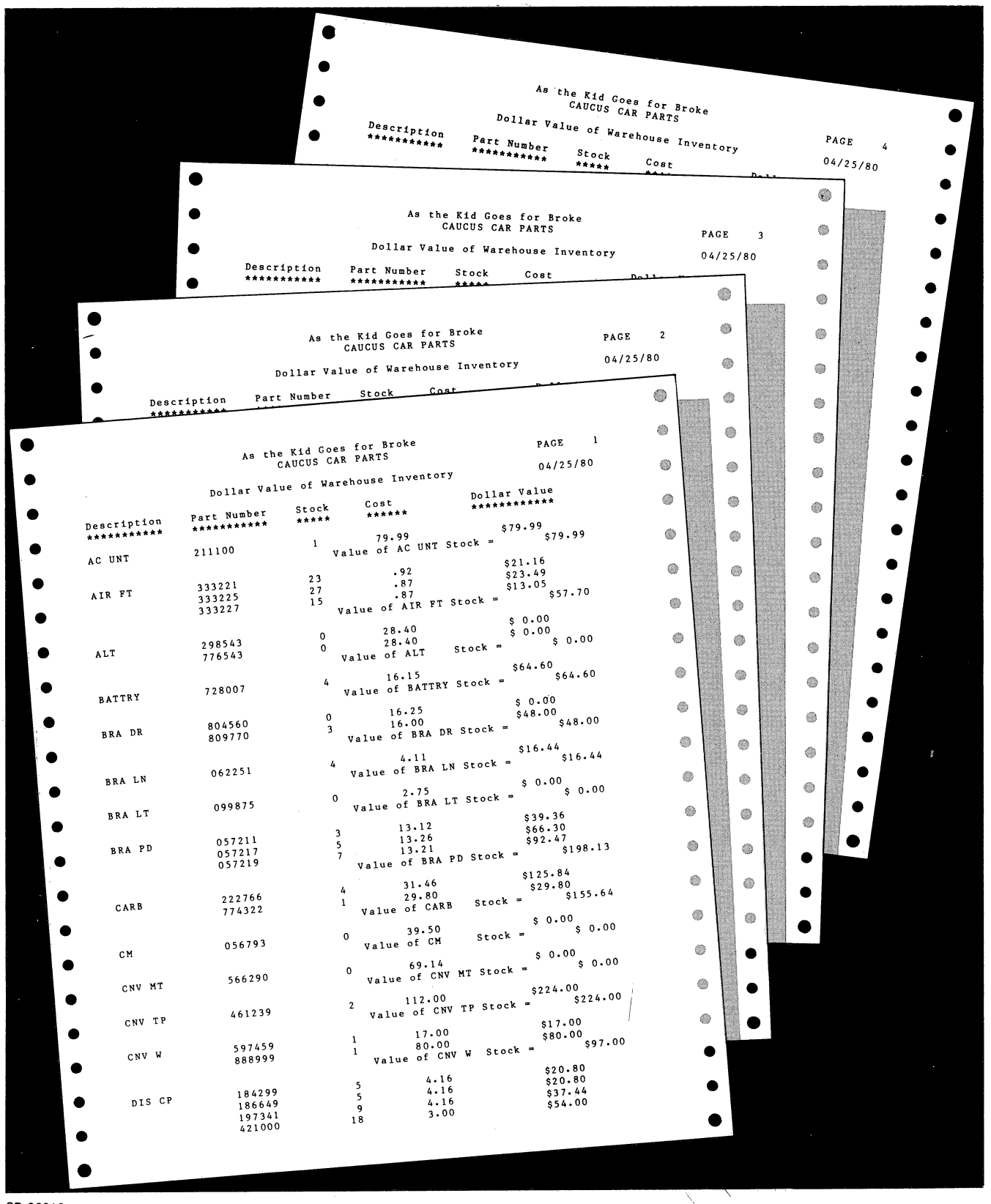
```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
>> RF STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
>> SELECT/RF ALL↵
No condition is specified
>> CONDITION ON HAND > 0↵
Condition is: ON HAND > 0
```

The series of commands used to create the report in Figure 5-2 were used to create the report shown in Figure 5-4. However, the report in Figure 5-4 used the OLDIES rformat.

The series of commands used to create the report in Figure 5-3 were used to create the report shown in Figure 5-5. However, the report in Figure 5-5 used the OLDIES rformat.

Summary

In this chapter we have given you the reference material you need to define rformats. We've also described how you create, verify, and use rformats. In addition, we've explained the RWCHECK process that interactively verifies an rformat's syntax and produces a sample page.



As the Kid Goes for Broke
CAUCUS CAR PARTS
Dollar Value of Warehouse Inventory
PAGE 4
04/25/80

Description ********** | Part Number ********** | Stock ***** | Cost ******

As the Kid Goes for Broke
CAUCUS CAR PARTS
Dollar Value of Warehouse Inventory
PAGE 3
04/25/80

Description ********** | Part Number ********** | Stock ***** | Cost

As the Kid Goes for Broke
CAUCUS CAR PARTS
Dollar Value of Warehouse Inventory
PAGE 2
04/25/80

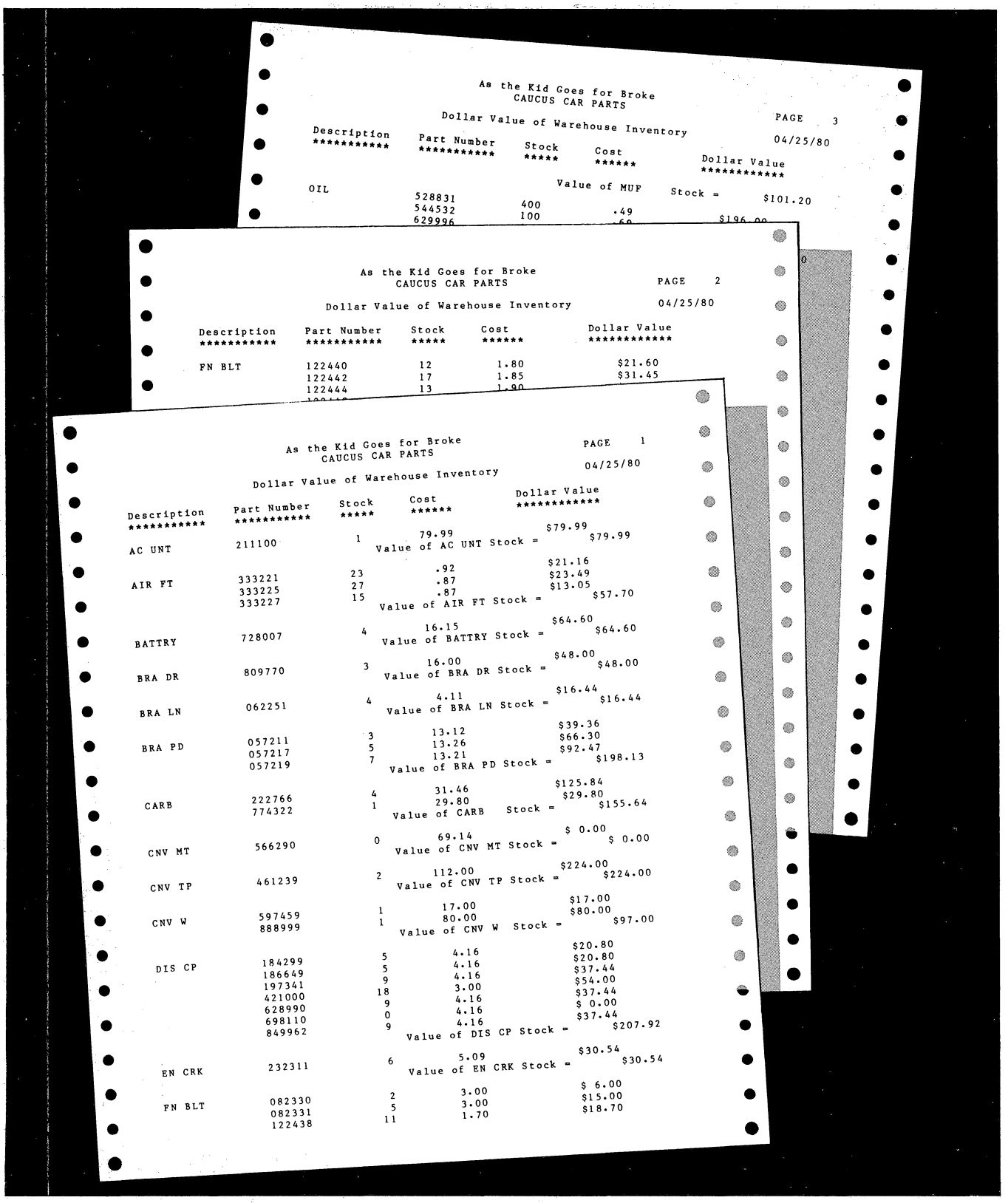
Description ********** | Part Number | Stock | Cost

As the Kid Goes for Broke
CAUCUS CAR PARTS
Dollar Value of Warehouse Inventory
PAGE 1
04/25/80

Description **********	Part Number **********	Stock *****	Cost ******	Dollar Value **********
AC UNT	211100	1	79.99	$79.99
		Value of AC UNT Stock =		$79.99
AIR FT	333221	23	.92	$21.16
	333225	27	.87	$23.49
	333227	15	.87	$13.05
		Value of AIR FT Stock =		$57.70
ALT	298543	0	28.40	$ 0.00
	776543	0	28.40	$ 0.00
		Value of ALT Stock =		$ 0.00
BATTRY	728007	4	16.15	$64.60
		Value of BATTRY Stock =		$64.60
BRA DR	804560	0	16.25	$ 0.00
	809770	3	16.00	$48.00
		Value of BRA DR Stock =		$48.00
BRA LN	062251	4	4.11	$16.44
		Value of BRA LN Stock =		$16.44
BRA LT	099875	0	2.75	$ 0.00
		Value of BRA LT Stock =		$ 0.00
BRA PD	057211	3	13.12	$39.36
	057217	5	13.26	$66.30
	057219	7	13.21	$92.47
		Value of BRA PD Stock =		$198.13
CARB	222766	4	31.46	$125.84
	774322	1	29.80	$29.80
		Value of CARB Stock =		$155.64
CM	056793	0	39.50	$ 0.00
		Value of CM Stock =		$ 0.00
CNV MT	566290	0	69.14	$ 0.00
		Value of CNV MT Stock =		$ 0.00
CNV TP	461239	2	112.00	$224.00
		Value of CNV TP Stock =		$224.00
CNV W	597459	1	17.00	$17.00
	888999	1	80.00	$80.00
		Value of CNV W Stock =		$97.00
DIS CP	184299	5	4.16	$20.80
	186649	5	4.16	$20.80
	197341	9	4.16	$37.44
	421000	18	3.00	$54.00

SD-02016

Figure 5-2. STOCK VALUE Report



As the Kid Goes for Broke
CAUCUS CAR PARTS

Dollar Value of Warehouse Inventory

PAGE 3
04/25/80

Description **********	Part Number **********	Stock *****	Cost ******	Dollar Value **********
OIL	528831		Value of MUF	Stock = $101.20
	544532	400	.49	
	629996	100	.60	$196.00

As the Kid Goes for Broke
CAUCUS CAR PARTS

Dollar Value of Warehouse Inventory

PAGE 2
04/25/80

Description **********	Part Number **********	Stock *****	Cost ******	Dollar Value **********
FN BLT	122440	12	1.80	$21.60
	122442	17	1.85	$31.45
	122444	13	1.90	

As the Kid Goes for Broke
CAUCUS CAR PARTS

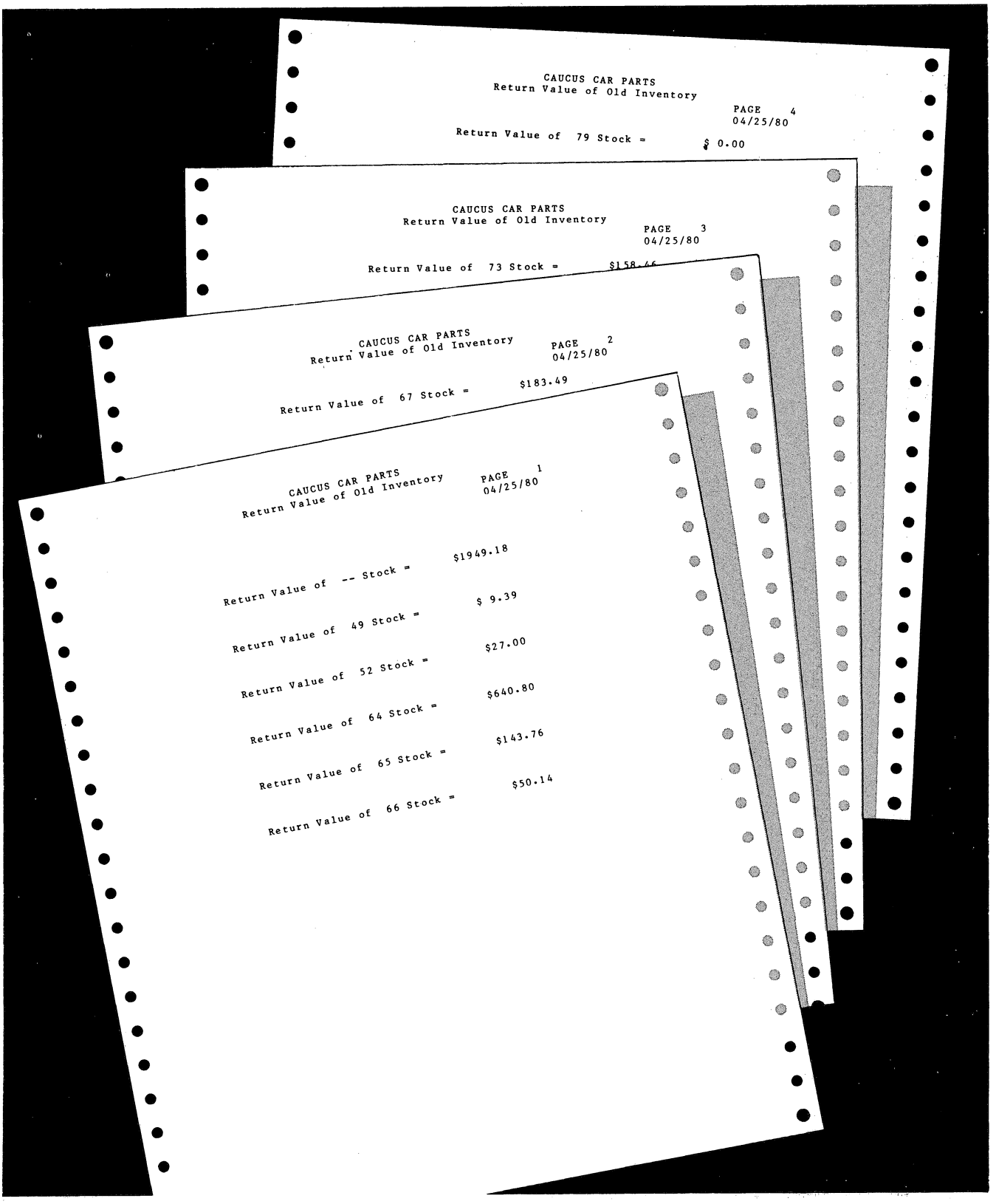
Dollar Value of Warehouse Inventory

PAGE 1
04/25/80

Description **********	Part Number **********	Stock *****	Cost ******	Dollar Value **********
AC UNT	211100	1	79.99	$79.99
			Value of AC UNT Stock =	$79.99
AIR FT	333221	23	.92	$21.16
	333225	27	.87	$23.49
	333227	15	.87	$13.05
			Value of AIR FT Stock =	$57.70
BATTRY	728007	4	16.15	$64.60
			Value of BATTRY Stock =	$64.60
BRA DR	809770	3	16.00	$48.00
			Value of BRA DR Stock =	$48.00
BRA LN	062251	4	4.11	$16.44
			Value of BRA LN Stock =	$16.44
BRA PD	057211	3	13.12	$39.36
	057217	5	13.26	$66.30
	057219	7	13.21	$92.47
			Value of BRA PD Stock =	$198.13
CARB	222766	4	31.46	$125.84
	774322	1	29.80	$29.80
			Value of CARB Stock =	$155.64
CNV MT	566290	0	69.14	$ 0.00
			Value of CNV MT Stock =	$ 0.00
CNV TP	461239	2	112.00	$224.00
			Value of CNV TP Stock =	$224.00
CNV W	597459	1	17.00	$17.00
	888999	1	80.00	$80.00
			Value of CNV W Stock =	$97.00
DIS CP	184299	5	4.16	$20.80
	186649	5	4.16	$20.80
	197341	9	4.16	$37.44
	421000	18	3.00	$54.00
	628990	9	4.16	$37.44
	698110	0	4.16	$ 0.00
	849962	9	4.16	$37.44
			Value of DIS CP Stock =	$207.92
EN CRK	232311	6	5.09	$30.54
			Value of EN CRK Stock =	$30.54
FN BLT	082330	2	3.00	$ 6.00
	082331	5	3.00	$15.00
	122438	11	1.70	$18.70

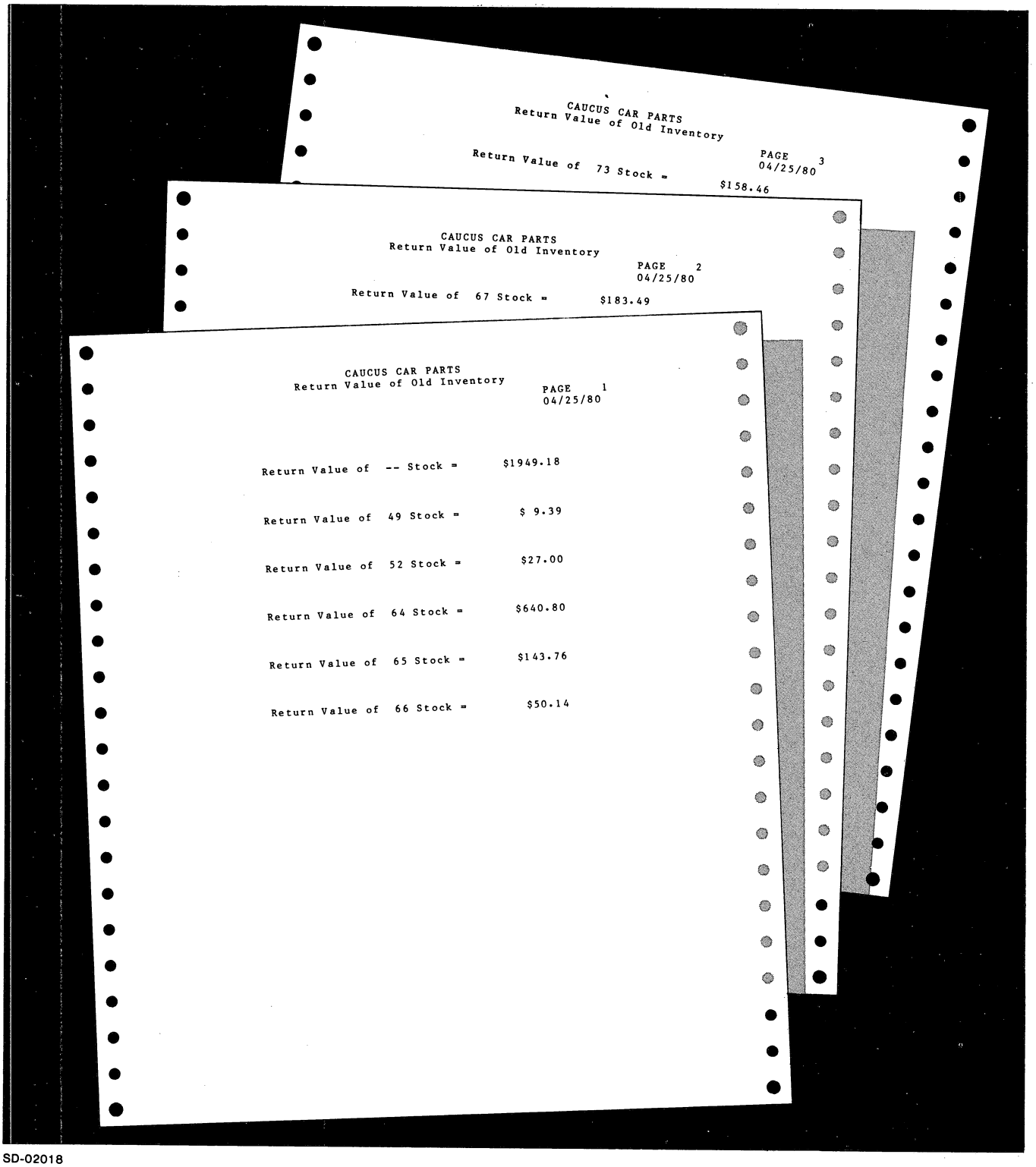
SD-02017

Figure 5-3. Report Created with the SELECT Command



SD-02018

Figure 5-4. An OLDIES Report Using a READ Command



SD-02018

Figure 5-5. An OLDIES Report Using a SELECT Command

End of Chapter

Chapter 6
Command Reference Section

In this chapter, we describe the 15 Query commands and the CLI command used to invoke the Query, with their arguments and switches. You should use this chapter for reference; if you have a question about the syntax or function of any command or switch, you can easily find the answer in this chapter. If you want a working description of a command, refer to the appropriate sections in Chapter 2.

We've arranged the commands in alphabetical order for fast reference with one exception -- the CLI command to invoke the Query (QUERY) appears first.

The commands described in this chapter, and the order they appear, are

QUERY
BYE
CLI
CONDITION
DOWN
HELP
KEY
LEVELMARK
NEXT
PATH
PRIOR
QFORMAT
READ
RFORMAT
SELECT
UP

We organized each command description so you can quickly find the information you need with the headings as follows:

Command	Lists the command's name and briefly tells you its function.
Syntax	Shows you the command syntax, including arguments and switches.
Switches	Lists and explains the function of any command switch.
Description	Describes the function of the command, and explains how to use it in a Query session.
Example & Explanation	Gives you an actual Query example using the commands and fully explains the example.

The only exception to this arrangement is the QUERY command. Since this manual is all about using the Query, we are not including a description of the command or an example.

QUERY
Invokes the Query from the CLI.

Syntax

QUERY [{ /B=*filename* | /T=*filename* | /Q=*filename* }] *[name]*

Where:

name is the INFOS file you wish to query.

Switches

/B=filename	Identifies the batch input filename from which the Query will get its commands.
/T=filename	Creates a trail file to which all your commands and the system's responses will be sent.
/Q=filename	Creates a Query trail file to which all your commands will be sent. You can use the Query trail file later as a macro or batch input file.

BYE

Ends a Query session.

Syntax

BYE

Switches

None.

Description

The BYE command terminates a Query session. Use this command to close the INFOS file you have queried and update any outstanding list files. BYE returns you to the AOS CLI.

Example

Type BYE, leave the Query, and return to the CLI.

```
>> BYE↵

)
```

CLI

Creates a subordinate CLI.

Syntax

CLI

Switches

None.

Description

The CLI command creates a subordinate CLI process (called a *son).* The CLI son enables you to work in the CLI without ending the Query session. You cannot use the CLI command in a Query macro. If you cannot create a CLI son with the CLI command, check with your system manager to find out whether you are privileged for this action. Type BYE to return to the Query from a CLI son.

If you use this command to return to the CLI during a Query session, any list files or text files to which you wrote data using the /L or /L=filename switches won't be closed yet. In addition, you cannot change @ LIST or your search list from the CLI son process.

Example

You type CLI; the Query then creates a son process.

```
>> CLI↵

AOS CLI REV 3.03    9-FEB-80        10:02:34
)
```

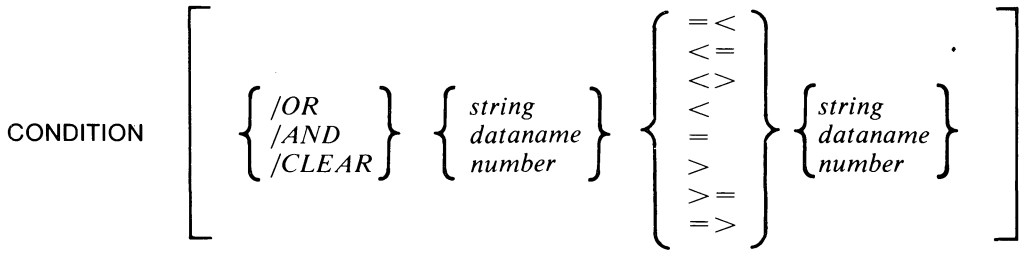
When you're ready to return to your Query session, type BYE, and the son process terminates:

```
) BYE↵
AOS CLI    TERMINATING    9-FEB-80    10:10:21
> >
```

CONDITION

Sets or displays selection criteria.

Syntax



Where:

string	is a string of ASCII characters, enclosed in either single or double quotation marks.
dataname	is the name of a data item found in the current qformat.
number	is a string of characters that can at minimum consist of digits (0 thru 9). In addition, it can include a sign (- or +) and/or a decimal point (.).
=	equal to
< >	not equal to
>	greater than
<	less than
= >	equal to or greater than
= <	equal to or less than
> =	greater than or equal to
< =	less than or equal to

Switches

/AND	Logically AND this relation to the current condition.
/OR	Logically OR this relation to the current condition.
/CLEAR	Clears the current condition.

Description

You use the CONDITION command to set or display a selection criteria the SELECT command will use. With no argument, CONDITION displays the currently defined condition. The selection criteria depends upon the currently defined qformat which identifies datanames and data types. When you establish a new qformat, the condition is no longer defined.

The argument you use with the CONDITION command creates a simple relation between datanames and/or constants. Once you have established one relation, you can append others to it using the /AND or /OR switch. You can append up to 15 conditions to your original condition statement. In most cases, if you include both /AND and /OR conditions, you should put /OR first because the Query evaluates the expression in a left to right order.

Examples

You log on to the Query and type CONDITION:

```
INFOS Query / Report Writer .... At Your Service
>> CONDITION↵
No condition is specified
```

You try to set a CONDITION before activating a qformat:

```
>> CONDITION ON HAND <= 10↵
There is no current qformat selected
```

You activate a qformat, then set a CONDITION:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> CONDITION ON HAND <= 10↵
Condition is: ON HAND < = 10
```

CONDITION (continued)

You then append a second and third CONDITION:

```
>> CONDITION/AND YEAR = 80↓
Condition is: ON HAND < = 10
        AND   YEAR = 80
>> CONDITION/OR YEAR = 79↓
Condition is: ON HAND < = 10
        AND   YEAR = 80
        OR    YEAR = 79
```

You move yourself in the index to a position where you can SELECT records, and then SELECT:

```
>> PATH PARTS↓
Path is PARTS
>> DOWN↓
Path is PARTS:056793
>> SELECT/COUNT/RETURN ALL↓
        1 records selected
Path is PARTS:056793
```

You unsuccessfully try to change the CONDITION; the Query tells you how to do it correctly:

```
>> CONDITION YEAR = 78↓
You must specify either /AND or /OR
```

You clear the current CONDITION and start over again:

```
>> CONDITION/CLEAR↓
No condition is specified
>> CONDITION ON HAND <= 10↓
Condition is: ON HAND < = 10
>> SELE/COUNT/RETURN ALL↓
        53 records selected
Path is PARTS:056793
```

You append a CONDITION and SELECT:

```
>> CONDITION/AND YEAR = 66↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
>> SEL/RETURN/COUNT ALL↓
        50 records have been scanned
        1 records selected
Path is PARTS:056793
```

You append another CONDITION and SELECT. Then again, and again, until you have a total of seven CONDITIONS:

```
>> CONDITION/AND MAKE = "FORD"↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
        AND   MAKE = "FORD"
>> SELECT/COUNT/RETURN ALL↓
        50 records have been scanned so far
        NO records selected
Path is PARTS:056793
>> CONDITION/OR YEAR = 67↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
        AND   MAKE = "FORD"
        OR    YEAR = 67
>> SELE/RETURN/COUNT ALL↓
        2 records selected
Path is PARTS:056793
>> CONDITION/OR YEAR = 68↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
        AND   MAKE = "FORD"
        OR    YEAR = 67
        OR    YEAR = 68
>> SEL/RETURN/COUNT ALL↓
        3 records selected
Path is PARTS:056793
>> CONDITION/OR MAKE = "MG"↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
        AND   MAKE = "FORD"
        OR    YEAR = 67
        OR    YEAR = 68
        AND   MAKE = "FORD"
        OR    MAKE = "MG"
>> SEL/RETURN/COUNT ALL↓
        4 records selected
Path is PARTS:056793
>> CON/OR MAKE = "VOLVO"↓
Condition is: ON HAND < = 10
        AND   YEAR = 66
        AND   MAKE = "FORD"
        OR    YEAR = 67
        OR    YEAR = 68
        OR    MAKE = "MG"
        OR    MAKE = "VOLVO"
>> SEL/RETURN/COUNT ALL↓
        7 records selected
Path is PARTS:056793
```

DOWN

Descends one subindex level.

Syntax

DOWN *//SUBINDEXES]*

Switch

/SUBINDEXES	Flags the presence of a subindex beneath the retrieved key.

Description

You use the DOWN command to move down (from one subindex level to a lower one) in your file. The DOWN command sets the current path by descending one subindex level; it also retrieves the first key in the subindex and appends it to the path. The Query then displays the current path.

If you try to do a DOWN when located above an empty subindex, the Query will not retreive a key and the path will remain unchanged. To find out if any subindexes exist beneath the retreived key, use the /SUBINDEX switch with the DOWN command.

Example

You log on to the Query, and move DOWN as far as you can go:

```
INFOS Query / Report Writer .... At Your Service
>> PATH↵
Path is above the index
>> DOWN↵
Path is PARTS
>> DOWN↵
Path is PARTS:056793
>> DOWN↵
Oops ! No subindex under the current path
```

You go DOWN, checking to see whether you have subindexes below. The Query returns a message, indicating a lower subindex.

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN/SUBINDEXES↵
Path is PARTS BY TYPE:AC UNT
*** Subindex beneath this entry
```

You move to another index and go DOWN, checking for subindexes below. The Query does not return a message, so you know that there are no lower subindexes.

```
>> PATH PO↵
Path is PO
>> DOWN/SUBINDEXES↵
Path is PO:770945
```

HELP

Provides information about the Query.

Syntax

HELP [{ /L | /L=*filename* }] *[command]*

Where:

command is the topic for which you're seeking help.

Switches

/L Directs output to @ LIST.

/L=filename Directs output to the named file.

Description

The HELP command provides information about all the Query commands. You can request general help, reference help for commands (including the command's purpose, syntax, and switches), and examples of commands.

If you use either switch, the help message will be sent to a text file, which you can then print. Use the command name for which you want help as an argument.

Example

To find out what help is available, type HELP without an argument:

```
>> HELP↓
Help is available on the following topics :

KEY                 QFORMS_FILE         CLI                 SELECT
PATH                CONDITION           BYE                 INDEX
RFORMS_SYNTAX       HELP                LEVELMARK           EX_LEVELMARK
PRIOR               DOWN                READ                NEXT
UP                  EX_UP               EX_PRIOR            QFORMS_SYNTAX
EX_DOWN             EX_READ             EX_KEY              QFORMS_CREATING
QFORMS_NONASCII     QFORMS_ERRORS       OVERVIEW            EX_SELECT
EX_PATH             EX_CONDITION        RFORMAT             EX_RFORMAT
EX_NEXT             QFORMAT             EX_QFORMAT

If you want more help, type

>> HELP HELP↓
HELP
        Provides information about the Query.

Syntax  HELP[switch] [command]

Switches  /L            Directs output to @LIST.
          /L=filename   Directs output to the named file.

Use     The HELP command provides information about all of the Query commands for your convenience at the console.
        The HELP messages include the command's purpose, format, examples, and switches.

    For the list of available HELP topics, type:
        >> HELP
```

You want some help with the KEY command:

```
>> HELP KEY )
KEY

        Searches the current index level.

Syntax  KEY[/APPROX]  key

Switch  /APPROX       Does an appropriate search instead of a generic search (i.e., searches for the next key equal to or
                      greater than the key you specified).

Use     The KEY command changes the current path by peforming a keyed search in the current index level. KEY takes a
        key fragment as an argument and does a generic keyed search for that fragment in the subindex. If the search
        succeeds, then the path will be changed.

    For an example of the KEY command, type:
        >> HELP EX_KEY
```

That's not enough help. You need an example, so you type

```
>> HELP EX_KEY )
Example for KEY                          Explanation

>> PATH PARTS BY TYPE                    You move to a subindex,
Path is PARTS BY TYPE
>> DOWN                                  and descend to the next level.
Path is PARTS BY TYPE:AC UNT
>> KEY OIL                               You then KEY to a desired position
Path is PARTS BY TYPE:OIL                in the new subindex,
>> KEY/APPROX D                          and then KEY to the position
Path is PARTS BY TYPE:DIS CP             nearest to another desired place.
>> DOWN                                  You go DOWN again,
Path is PARTS BY TYPE:DIS CP:184299
>> KEY 8                                 and then KEY to a new subindex range.
Path is PARTS BY TYPE:DIS CP:849962
>> KEY ZZZ                               Now you KEY to another range,
Keyed search was unsuccessful            but your search fails.
```

KEY

Searches the current index level.

Syntax

KEY *[/APPROX] [name]*

Where:

name is a key fragment.

Switch

/APPROX Does an approximate search instead of a generic search (i.e., searches for the next key equal to or greater than the key you specified).

Description

The KEY command changes the current path by performing a keyed search in the current index level.

KEY takes a key fragment as an argument and does a generic keyed search for that fragment in the subindex. If the search succeeds, then the path changes.

Example

You move to a subindex, and descend to the next level:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
```

You then try to KEY to a desired position in the subindex:

```
>> KEY TII↵
Keyed search was unsucessful
```

Your search was unsuccessful. You try again, using the same key with the /APPROX switch:

```
>> KEY/APPROX TII↵
Path is PARTS BY TYPE:TL PIP
```

This time you found a new key.

You move to a new index and try a different KEY search:

```
>> PATH PARTS↵
Path is PARTS
>> DOWN↵
Path is PARTS:056793
>> KEY 091↵
Keyed search was unsucessful
>> KEY/APPROX 091↵
Path is PARTS:099875
```

LEVELMARK

Sets or displays the current level separating character.

Syntax

LEVELMARK *[c]*

Where:

c represents any keyboard character.

Switches

None.

Description

You use the LEVELMARK command to set or display the current level separating character. By default, the colon (:) separates keys from different levels of the path. In some cases, however, you may not want to use this character. Therefore, you use the LEVELMARK command to change the key separator to a different character. With no argument, LEVELMARK displays the current levelmark character. You specify a different character as an argument in order to change the levelmark character. The key level separator must be a single character.

Example

You ask for the currently defined LEVELMARK character and the Query displays it:

```
>> LEVELMARK↵
Level Separator is :
>> PATH↵
Path is PARTS:056793
```

You change the LEVELMARK character and the Query verifies it:

```
>> LEVELMARK 9↵
Level Separator is 9
```

When you do a PATH command, you see that this new LEVELMARK character causes confusion:

```
>> PATH↵
Path is PARTS9056793
```

You decide to change the LEVELMARK one more time, the Query verifies it, and you try to move to another index:

```
>> LEVELMARK #↵
Level Separator is #
>> PATH PARTS BY TYPE:AC UNT↵
Path is PARTS # 056793
```

The PATH command did not work because you did not use the new LEVELMARK character; your PATH remains unchanged. You try again, using the new LEVELMARK character:

```
>> PATH PARTS BY TYPE#AC UNT↵
Path is PARTS BY TYPE # AC UNT
```

NEXT

Positions forward in the current index level.

Syntax

NEXT *//SUBINDEXES] [n]*

Where:

n is an integer.

Switch

/SUBINDEXES Flags the presence of a subindex beneath the retrieved key.

Description

You use the NEXT command to change the current path by moving forward in the current index. With no argument, NEXT positions the path one key forward, outputs the retrieved key, and then prints the new path. You can use an integer as an argument to the NEXT command; the Query then moves forward that number of positions.

Example

You move to a subindex, and then move to the NEXT key:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> NEXT↵
Path is PO
```

Move DOWN, and check the NEXT key:

```
>> DOWN↵
Path is PO:770945
>> NEXT↵
Path is PO:780011
```

Now, check the NEXT 3 keys:

```
>> NEXT 3↵
        780019
        780021
Path is PO:780031
```

Check if the NEXT 3 keys have subindexes:

```
>> NEXT/SUB 3↵
        780034
        780044
Path is PO:780045
```

You received no message, so you know there are no subindexes below these keys. Now, move to another subindex:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
```

Move DOWN, and check the NEXT 3 keys:

```
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> NEXT 3↵
        AIR FT
        ALT
Path is PARTS BY TYPE:BATTRY
```

Check the NEXT 3 keys, looking for subindexes:

```
>> NEXT/SUB 3↵
        BRA DR
*** Subindex beneath this entry
        BRA LN
*** Subindex beneath this entry
Path is PARTS BY TYPE:BRA LT
*** Subindex beneath this entry
```

The system responds with the message that informs you there are subindexes beneath these keys.

PATH

Sets or displays the current key path.

Syntax

PATH *[key] [:key ...]*

Where:

key is a series of key fragments.

Switches

None.

Description

You use the PATH command to set or display the current key path. With no argument, PATH displays the current path. You can use a series of key fragments, separated by a colon (:), or the current levelmark, as an argument to the PATH command; the Query then performs a generic keyed search on that multilevel key. If the search succeeds, the Query changes the current path to the one specified. If the search fails, however, the Query positions the path as far down in the index structure as it can and still successfully recover a key, except if you have exceeded the maximum number of subindex levels.

Example

You log on to the Query and check your PATH:

INFOS Query / Report Writer .... At Your Service
>> PATH↵
Path is above the index

You PATH to a subindex:

>> PATH PARTS↵
Path is PARTS

You try to PATH to another subindex, but fail:

>> PATH parts by type↵
Path is PARTS

You did not type the name of the subindex in uppercase letters, so the Query did not recognize it. Your PATH is unchanged. You try again:

>> PATH PARTS BY TYPE:O↵
Path is PARTS BY TYPE:OIL

You specify yet another PATH, but the Query can't find the specified key. The PATH changes to the last successfully recovered key:

>> PATH PARTS BY TYPE:AC UNT:0↵
Path is PARTS BY TYPE:AC UNT

You specify another PATH, but this one has more index levels than your INFOS file allows:

>> PATH PARTS BY TYPE:OIL:STP:10W↵
Path has too many levels

PRIOR

Positions backward in the current index level.

Syntax

PRIOR *[/SUBINDEXES] [n]*

Where:

n is an integer.

Switch

/SUBINDEXES	Flags the presence of a subindex beneath the retrieved key.

Description

You use the PRIOR command to change the current path by positioning backward in the current index. Without an argument, PRIOR moves one key backward, outputs the retrieved key, and then prints the new path. You can use an integer as an argument to the PRIOR command; the Query then moves backward that number of positions.

Example

You're in a subindex, and you move to the PRIOR key:

```
>> PATH↵
Path is PARTS BY TYPE:CM
>> PRIOR↵
Path is PARTS BY TYPE:CARB
```

You then move three keys PRIOR to your current position:

```
>> PRIOR 3↵
        BRA PD
        BRA LT
Path is PARTS BY TYPE:BRA LN
```

You go back to the original position and check if the four PRIOR keys have subindexes:

```
>>PATH PARTS BY TYPE:CM↵
Path is PARTS BY TYPE:CM
>> PRIOR/SUB 4↵
        CARB
*** Subindex beneath this entry
        BRA PD
*** Subindex beneath this entry
        BRA LT
*** Subindex beneath this entry
Path is PARTS BY TYPE:BRA LN
*** Subindex beneath this entry
```

You specify a faulty argument and the Query alerts you to your error:

```
>> PRIOR T↵
Argument must be an integer between 1 and 32767
```

You try a PRIOR command at a point in the index where no more entries exist:

```
>> PATH↵
Path is PARTS BY TYPE:BRA LN:062251
>> PRIOR↵
Alas - no more entries in this subindex
```

QFORMAT

Creates and manipulates qformats.

Syntax

QFORMAT [/CLEAR | /DISPLAY | /L | /L=*filename* | /SPEED | /LINEDIT | /NAMES] *[name]*

or

FORMAT [/CLEAR | /DISPLAY | /L | /L=*filename* | /SPEED | /LINEDIT | /NAMES] *[name]*

Where:

name is the name of a predefined qformat.

Switches

/CLEAR Clears the current qformat -- no argument allowed.

/DISPLAY Displays the current qformat as it appears in the .QFORMS file -- no argument allowed.

/L Directs the output to @ LIST.

/L=filename Appends the output *filename*.

/SPEED Calls up SPEED to modify the .QFORMS file -- no argument allowed.

/LINEDIT Calls up LINEDIT to modify the .QFORMS file -- no argument allowed.

/NAMES Displays the names of all existing qformats.

Description

The QFORMAT command enables you to create and manipulate qformats that describe your data records. You use these qformats for outputting records and formulating selection criteria. All your qformats are stored in a Query qformat file so you can retrieve them whenever you query an INFOS file.

The QFORMAT command with no switches or arguments prints the name of the currently active qformat. With a qformat name as an argument, QFORMAT changes the currently active qformat to the one named.

You'll use the /LINEDIT or /SPEED switches with the QFORMAT command when you want to modify an existing qformat, or when you want to create a new one. The /LINEDIT and /SPEED switches invoke the respective system text editors with the .QFORMS file as input. After you modify the .QFORMS file and exit from an editor, the Query will syntactically scan the .QFORMS file and return error messages if you have made any mistakes.

Users of Query Rev. 1 should note that the FORMAT command still works. This Rev. of the Query uses the QFORMAT command in addition to the FORMAT command; both commands perform the same function.

QFORMAT (continued)

Example

You ask which qformat is active and then look to see which ones are defined:

```
>> QFORMAT↵
There is no current qformat selected
>> QFORMAT/NAMES↵
The following qformats have been defined on this file :
        RETAIL
        PARTS
        OPEN PO'S
        TYR PO LOCATIONS
```

You activate the RETAIL qformat and tell the Query to display it:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> QFORMAT/DIS↵
PART NUM        1    6    ASCII
PRICE           60   65   ASCII
DESCRIPTION     7    12   ASCII
SUPPLIER        40   49   ASCII
YEAR            13   14   ASCII
MAKE            15   20   ASCII
MODEL           21   30   ASCII
ON HAND         31   33   ASCII
```

You deactivate the current qformat:

```
>> QFORMAT/CLEAR↵
There is no current qformat selected
```

READ

Reads the database record.

Syntax

READ [{ *L* / *L=filename* / */RETURN* / */UPTO [key]* / */TRAVERSE* / */RFORMAT* / */SCREEN* }] [{ *ALL* / *n* }]

Where:

n	is an integer.
key	is a key fragment.

Switches

/L	Directs output to @ LIST.
/L=filename	Directs output to *filename*.
/RETURN	On a multirecord READ, returns the path to its original position.
/UPTO	Tells the Query to READ all records up to a specified key. You must specify the key as an argument.
/TRAVERSE	Allows you to perform an extended search through a selector subindex structure.
/RFORMAT	Tells the Query to READ the record and output it according to the currently active rformat.
/SCREEN	Uses the special features of the DASHER Display terminal to display a report screen by screen.

Description

You use the READ command to read a database record. If you have activated a qformat and do not use an argument, the READ command reads the record at the current path according to the specified qformat. You can specify a positive integer as an argument to the READ comand; the Query then displays the number of records you specified. If you have not specified a qformat, the Query displays the data as it appears on the record in one straight line.

If you have activated an rformat and use the /RFORMAT switch, the Query will display the record in the report format defined by the rformat.

Example

You check your qformat, PATH, and then try to read a record:

```
>> QFORMAT↓
The current qformat is PARTS
>> PATH↓
Path is PARTS BY TYPE:AC UNT
>> READ↓
Key has no database record
```

You're PATH isn't correct, so you move DOWN, and try to READ 3 records:

```
>> DOWN↓
Path is  PARTS BY TYPE:AC UNT:211100
>> READ 3↓
```

```
PART NUM          : 211100
DESCRIPTION       : AC UNT
YEAR              : 75
MAKE              : AUDI
MODEL             : FOX
ON HAND           :   2
IN WAREHOUSE      :   1
REORDER           :   2
SUPPLIER          : POR AUDI
SUP ADDRESS       : FRNFT   WG
RETAIL PRICE      :  89.99
WHOLESALE PRICE   :  79.99
```

Although you asked the Query to READ 3, because you hit the end of the subindex it only READ one. Since you're in a selector subindex, you might try a /TRAVERSE switch:

```
>> READ/TRAVERSE 3↓
```

```
PART NUM          : 211100
DESCRIPTION       : AC UNT
YEAR              : 75
MAKE              : AUDI
MODEL             : FOX
ON HAND           : 2
IN WAREHOUSE      : 1
REORDER           : 2
SUPPLIER          : POR AUDI
SUP ADDRESS       : FRNFT WG
RETAIL PRICE      :  89.99
WHOLESALE PRICE   :  79.99
```

```
PART NUM          : 333221
DESCRIPTION       : AIR FT
YEAR              : 72
MAKE              : CHEV
MODEL             : IMPALA 225
ON HAND           : 12
IN WAREHOUSE      : 23
REORDER           : 40
SUPPLIER          : GEN MOTORS
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 1.95
WHOLESALE PRICE   :  .92
```

```
PART NUM          : 333225
DESCRIPTION       : AIR FT
YEAR              : 73
MAKE              : CHEV
MODEL             : BELAIR 225
ON HAND           : 17
IN WAREHOUSE      : 27
REORDER           : 40
SUPPLIER          : GEN MOTORS
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 1.89
WHOLESALE PRICE   :  .87
```

READ (continued)

You clear the qformat and repeat the last two READ commands, using the /RETURN switch to ensure that your PATH is the same at the beginning of each READ:

```
>> QF/CLEAR↓
There is no current qformat selected
>> READ/RETURN 3↓

211100AC UNT75AUDI  FOX          2  1  2POR AUDI  FRNFT   WG 89.99 79.99
Path is  PARTS BY TYPE:AC UNT:211100
>> READ/TRAVERSE/RETURN 3↓

211100AC UNT75AUDI  FOX          2  1  2POR AUDI  FRNFT   WG 89.99 79.99

333221AIR FT72CHEV  IMPALA 225 12 23 40GEN MOTORSDETROIT MI  1.95   .92

333225AIR FT73CHEV  BELAIR 225 17 27 40GEN MOTORSDETROIT MI  1.89   .87
Path is  PARTS BY TYPE:AC UNT:211100
```

You set an rformat and ask the Query to READ all the records and output them in an output file (called STOCK_TEST). By using the /RFORMAT switch, you're telling the Query to organize these records into the report defined by the current rformat. To see the results of this command (the contents of STOCK_TEST), either try the example yourself or go back to Chapter 5 where we discuss rformats and the resulting reports.

```
>> RF STOCK VALUE↓
The current qformat is PARTS
The current rformat is STOCK VALUE
>> READ/RETURN/TRAVERSE/RF/L=STOCK_TEST ALL↓
      4 pages written into the list file
Path is PARTS BY TYPE:AC UNT:211100
```

Change the qformat and use the /UPTO switch to a the Query to READ as many records as it can until i encounters the specified key fragment:

```
>>QFORMAT RETAIL↓
The current qformat is RETAIL
>>PATH↓
Path is PARTS BY TYPE:AC UNT:211100
>> READ/TRAVERSE/RETURN/UPTO 28↓

PART NUM     : 211100
PRICE        : 89.99
DESCRIPTION  : AC UNT
SUPPLIER     : POR AUDI
YEAR         : 75
MAKE         : AUDI
MODEL        : FOX
ON HAND      : 2

PART NUM     : 333221
PRICE        : 1.95
DESCRIPTION  : AIR FT
YEAR         : 72
MAKE         : CHEV
MODEL        : IMPALA 225
ON HAND      : 12
Path is PARTS BY TYPE:AC UNT:211100
```

RFORMAT

Creates and manipulates rformats.

Syntax

RFORMAT [{ /CLEAR | /DISPLAY | /L | /L=filename | /LINEDIT | /SPEED | /NAMES }] *[name]*

Where:

name is the name of a predefined rformat.

Switches

/CLEAR Deactivates the current rformat -- no argument allowed.

/DISPLAY Displays the current rformat -- no argument allowed.

/L Directs output to @ LIST.

/L=filename Appends the output to *filename.*

/LINEDIT Calls up LINEDIT to modify the .RFORMS file -- no argument allowed.

/SPEED Calls up SPEED to modify the .RFORMS file -- no argument allowed.

/NAMES Displays the names of all existing rformats.

Description

The RFORMAT command enables you to create and manipulate rformats. You use rformats for outputting data in the form of reports. All your rformats are stored in a Query rformat file.

The RFORMAT command without switches or arguments prints the name of the currently active rformat. With an rformat name as an argument, RFORMAT changes the currently active rfomrat to the one named.

You'll use the /LINEDIT or /SPEED switches with the RFORMAT command when you want to modify an existing rformat, or when you want to create a new one.

The /LINEDIT or /SPEED switches invoke the respective text editors with the .RFORMS file as input.

Example

You ask which rformat is active and then look to see which ones are defined:

```
>> RFORMAT↵
There is no current rformat selected
>> RFORMAT/NAMES↵
The following rformats have been defined:
        STOCK VALUE
        OLDIES
```

RFORMAT (continued)

Example (continued)

You activate the OLDIES rformat and tell the Query to display it:

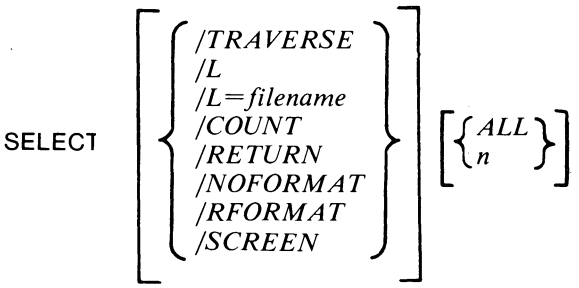
```
>> RFORMAT OLDIES↵
The current qformat is PARTS
The current rformat is OLDIES
>> RF/DIS↵
START_REPORT OLDIES
!           A REPORT FOR MANAGEMENT TELLING HOW MUCH REVENUE
!           CAN BE GENERATED BY RETURNING PARTS THAT ARE
!           PRE 1968
HEADER      1           C                       "CAUCUS CAR PARTS"
HEADER      2           C                       "Return Value of Old Inventory"
HEADER      3           60                      PAGE
HEADER      4           60                      DATE
QFORMAT     PARTS
LIN/PG      30
COL/LIN     80
DEFINE      DOLLAR VALUE                        ON HAND * WHOLESALE PRICE
DEFINE      RETURN VALUE                        90 % DOLLAR VALUE
PICTURE     RETURN VALUE                        $$$$$99V.99
SORT        YEAR
BREAK       YEAR    PRE_BREAK_SPACE
BREAK       YEAR    PRE_BREAK_SPACE
BREAK       YEAR    PRE_BREAK_SPACE
BREAK       YEAR    21                          "Return Value of"
BREAK       YEAR    38                          YEAR
BREAK       YEAR    41                          "Stock = "
BREAK       YEAR    52                          TOTAL ( RETURN VALUE )
END_REPORT
```

You deactivate the current rformat:

```
>> RFORMAT/CLEAR↵
There is no current rformat selected
```

SELECT
Searches for matching records.

Syntax



Where:

n is an integer.

Switches

/TRAVERSE Allows you to perform an extended search through a selector subindex structure.

/COUNT Counts, rather than displays, the number of matching records.

/L Directs output to @ LIST.

/L=filename Directs output to *filename.*

/RETURN On a multirecord SELECT, does not change the path from its original position.

/NOFORMAT Tells the Query to display the selected record as it appears in the database; on one straight line.

/RFORMAT Tells the Query to display the selected record according to the currently activated rformat.

/SCREEN Uses the special features of the DASHER Display terminal to display a report screen by screen.

Description

Use the SELECT command to search through the current subindex for records that conform to your selection criteria. The search begins at the current path and proceeds forward. When the Query finds a matching record, it outputs the record according to the currently active qformat. Without an argument, SELECT outputs the first matching record. You can use an integer or ALL as an argument to the SELECT command; the Query then selects and displays that number of records. If you use the /RFORMAT switch, the Query displays the record in the report format defined by the currently active rformat.

Example

You activate a qformat and ask for the current CONDITION:

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
>> CONDITION↵
No condition is specified
```

Finding no CONDITION specified, you create one:

```
>> CON YEAR = 78↵
Condition is: YEAR = 78
```

SELECT (continued)

After positioning the PATH at the correct level, ask the Query to SELECT ALL matching records, and display them as they appear in the database (by using the /NOFORMAT switch):

```
>>PATH↵
Path is  PARTS
>> DOWN↵
Path is  PARTS:056793
>> SEL/NOFORMAT ALL↵
249321RD TYR78FIRESTFB  165-13 90 70 80FIRESTONE DETROIT MI 41.23 32.20
804560BRA DR78RENAULLE CAR      0  0  6RENAULT USFT. LEE NJ 19.50 16.25
```

Change the condition and ask the Query to COUNT all the matching records:

```
>>CONDITION/CLEAR↵
No condition is specified
>> CONDITION YEAR = 80↵
Condition is: YEAR = 80
>> CONDITION/OR YEAR = 79↵
Condition is: YEAR = 80
      OR   YEAR = 79
>>PATH↵
Path is PARTS:056793
>> SELECT/COUNT/RETURN ALL↵
      1 records selected
Path is PARTS:056793
```

Now, activate an rformat and repeat the last SELECT command, asking the Query to send the matching records to an output file and to organize them in the form of the report defined by the active rformat. Note that you must declare the CONDITION again; the rformat uses a different qformat and by changing the qformat the CONDITION was automatically cleared.

```
>> RF STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
>> CONDITION YEAR = 80↵
Condition is: YEAR = 80
>> CONDITION/OR YEAR = 79↵
Condition is: YEAR = 80
      OR   YEAR = 79
>> SELECT/RF/L=SELECT_TEST/RETURN ALL↵
      1 pages written into the list file
Path is PARTS:056793
```

To see this report, either try the example yourself or refer to Chapter 5, where rformats and the reports they generate are fully discussed.

PATH to a different subindex and change the CONDITION. Then try to count the matching records:

```
>> PATH PARTS BY TYPE↵
Path is PARTS BY TYPE
>> DOWN↵
Path is PARTS BY TYPE:AC UNT
>> DOWN↵
Path is PARTS BY TYPE:AC UNT:211100
>> CONDITION/CL↵
No condition is specified
>> CONDITION MAKE = "FORD"↵
Condition is: MAKE = "FORD"
>> SELECT/COUNT/RETURN ALL↵
NO records selected
Path is PARTS BY TYPE:AC UNT:211100
```

Because you're in a selector subindex, Query hit th end of the subindex before it found any matchin records. Try again, using the /TRAVERSE switch:

```
>> SELECT/COUNT/RETURN/TRAVERSE ALL↵
      15 records selected
Path is PARTS BY TYPE:AC UNT:211100
```

UP

Moves to the next higher subindex level.

Syntax

UP *[n]*

Where:

n is an integer.

Switches

None.

Description

You use the UP command to move the current path to the next higher subindex level. Without an argument, UP moves upward and removes the last key from the current path. You can use an integer argument with the UP command; the Query then moves upward the specified number of positions.

Example

You check your PATH and decide to move UP three levels:

```
>> PATH↓
Path is PARTS BY TYPE:AC UNT:211100
>> UP 3↓
        AC UNT
        PARTS BY TYPE
Path is above the index
```

You try to move UP again, but you can't go any higher:

```
>> UP↓
Sorry, you are already at the top
```

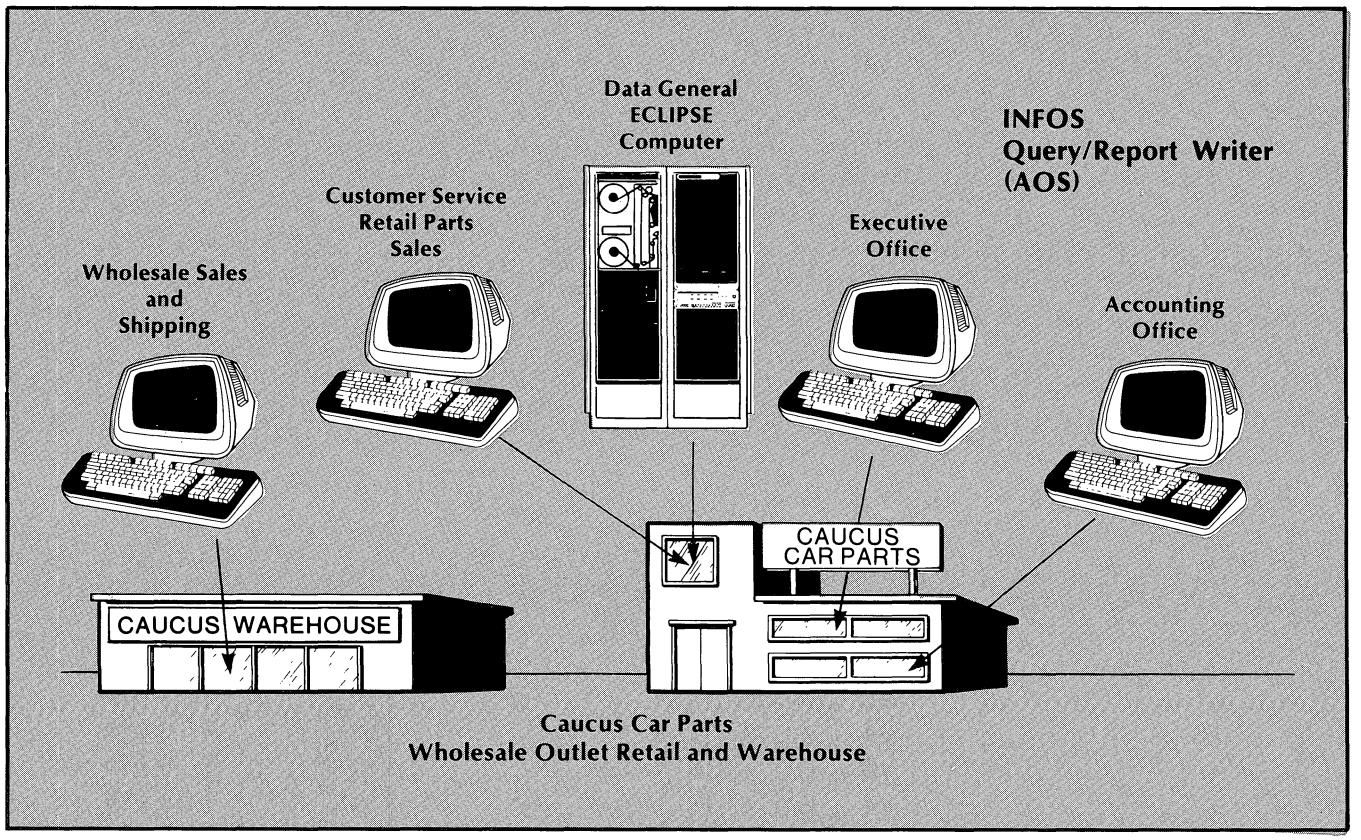
End of Chapter

Chapter 7
A Day in the Life of Caucus Car Parts

Throughout this manual we have used Caucus Car Parts as our sample business application. This chapter will demonstrate some everyday business uses of the INFOS Query/Report Writer system by showing a day in the life of Caucus Car Parts.

Background

Caucus Car Parts is a major distributor of American and imported car parts and accessories. Founded in 1953 by J.J. Caucus, the company has grown steadily over the years. Recently, it merged with Harris Auto Parts Co., another distributor of car parts and accessories. Because it sells to both wholesale and retail customers, Caucus maintains a huge inventory of over 50,000 items in a retail store and a nearby warehouse. The retail store, which also houses the company's offices, stocks fast-moving parts and accessories. The warehouse stocks slow-moving parts, duplicate stock from the retail store, oversized items such as tires, and miscellaneous surplus goods like oil. Figure 7-1 illustrates Caucus's business set-up.



SD-01335

Figure 7-1. Caucus Car Parts' Query Set-up

Before acquiring the INFOS system with the Query facility, Caucus, like many companies with data retrieval problems, relied entirely on a conventional paper record inventory and access system. For much of the company's history, this system worked adequately. But in recent years, the number of stock items, the variety of accessories, and the new market competition forced Caucus to review its record keeping arrangements. Caucus was in trouble. The company needed help with such problems as lost files, overstocked and undersold merchandise, oceans of paper and an overworked staff. Caucus needed a system that could solve the problems of the present and help achieve future growth.

Luckily for Caucus, Harris Auto Supply Co. was already using the INFOS system with the Query facility. When the two companies merged, Caucus followed the precedent set at Harris and converted their paper records to the new electronic medium. The two companies had carried similar stock and dealt with many of the same vendors, so the conversion was simple. The new Report Writer features were a welcome addition to both sides of the merger.

When Harris built their INFOS database, the particular index structure chosen made their INFOS system accessible to all company's employees, some of whom had no data processing background. The same was true for Caucus, and the original database structure stayed after the merger.

During the first few months that Caucus Car Parts worked with the Query facility, they organized the system so many of their routine Query operations could be performed using Query macros.

8:40 A.M.

By the time Virginia Spaulding arrives at work, the INFOS system is up and running. Virginia has been a retail counter sales clerk at Caucus for four years. She was suspicious of the Query at first. But after learning to work the terminal and seeing how it relieved some of the tedium of her job, she became an ardent enthusiast.

As Virginia stubbed out her first cigarette of the day, Lou Murphy, the parts manager at nearby Hastings Ford, walked in. "Hiya Ginny, I need a fuel pump for a '76 Mustang V-8 -- it's a dealer car, so if you have different makes, give me the cheapest one."

Virginia thought a second about the information she needed and how she could use the Query to retrieve it. As she walked over to her counter terminal she replied, "Let's see what's in stock." Virginia quickly invoked the Query from the CLI:

```
) QUERY CAUCUS↵
INFOS Query / Report Writer ...At Your Service
```

She used the PATH command to locate the fuel pump key in the PARTS BY TYPE subindex and went DOWN to the next subindex.

```
>> PATH PARTS BY TYPE:FP↵
Path is PARTS BY TYPE:FP
>> DOWN↵
Path is PARTS BY TYPE:FP    :596727
```

"Now," Virginia thought to herself. "I need to find the parts records for 76 Ford Mustang fuel pumps. I'm in the fuel pump subindex, so I'll need to specify two conditions. First, I'll activate the RETAIL qformat." She started typing.

```
>> QFORMAT RETAIL↵
The current qformat is RETAIL
```

"Now, I'll declare the two conditions."

```
>> CONDITION MODEL = "MUSTAN"↵
Condition is: MODEL = "MUSTAN"
>> CON/AND YEAR = 76↵
Condition is: MODEL = "MUSTAN"
        AND   YEAR = 76
```

"If I select all the records, well, that should do it."

```
>> SELECT ALL↵

PART NUM     : 596727
PRICE        : 16.00
DESCRIPTION  : FP
SUPPLIER     : ALLIED AP
YEAR         : 76
MAKE         : FORD
MODEL        : MUSTAN 350
ON HAND      : 3

PART NUM     : 596730
PRICE        : 23.87
DESCRIPTION  : FP
SUPPLIER     : FO MO CO
YEAR         : 76
MAKE         : FORD
MODEL        : MUSTAN 350
ON HAND      : 3

PART NUM     : 596730
PRICE        : 19.23
DESCRIPTION  : FP
SUPPLIER     : AMR PARTS
YEAR         : 76
MAKE         : FORD
MODEL        : MUSTAN 350
ON HAND      : 1
```

Virginia looked up from the terminal and smiled. "We've got three types Lou -- take your pick: an Allied for $16.00, one from AMR for $19.23, and a Ford for $23.87.

"Great, I'll take the Allied," Lou said without hesitation. "I'm also overhauling a Ford Granada back at the shop, same year. Would that fuel pump fit it?"

"Let's check it out." Virginia thought a moment, "If it's the same part, it'll have the same part number." And she began to type at the terminal.

"First, I'll clear the condition," she mumbled to herself. "Then I'll specify a new one. That will get me all the records that match the part number -- even if they're not the same make and model."

```
>> CONDITION/CLEAR↓
No condition is specified
>> CONDITION PART NUM = 596727↓
Condition is PART NUM = 596727
>> UP↓
Path is PARTS BY TYPE:FP
>> DOWN↓
Path is PARTS BY TYPE:FP    :596727
>> SELECT ALL↓

PART NUM : 596727
PRICE : 16.00
DESCRIPTION : FP
SUPPLIER : ALLIED AP
YEAR : 76
MAKE : FORD
MODEL : MUSTAN 350
ON HAND : 3
```

"No, it won't fit," Virginia said turning back to the counter. "Want me to find one that will?"

"Don't bother. Just give me the Allied Mustang pump. I'll get original equipment for the customer's car."

Virginia picked up the counter phone and called the stock room. "Hey Red, bring me an Allied 596727 -- that's a fuel pump."

As Virginia finished writing up the sales slip, the fuel pump arrived from the stock room. "Here Lou," she motioned. "Just sign here and you're pumping gas."

9:45 A.M.

A teenager approached the counter, nervously scanning the store. "Could you please tell me if you stock radial tires for Volvos," he asked softly. "The size is 165-13, any kind."

Virginia smiled at the request. "I can do better than that," she said, and headed back to her terminal. "I'll tell you what *kinds* of radials, what *brands* we have in stock, and what the *prices* are." Within seconds Virginia had formulated her command sequence.

"Radial tires," she mused. "I'll move to PARTS BY TYPE and go down a couple of levels to radials."

```
>> PATH PARTS BY TYPE:R↓
Path is PARTS BY TYPE:RD TYR
>> DOWN↓
Path is PARTS BY TYPE:RD TYR:123769
```

"I'll specify two conditions -- one for steel and another for fiber belt.

```
>> QFORMAT RETAIL↓
The current qformat is RETAIL
>> CONDITION MODEL="STL 165-13"↓
Condition is:    MODEL="STL 165-13"
>> CONDITION/OR MODEL="FB 165-13"↓
Condition is: MODEL="STL 165-13"
        OR    MODEL="FB 165-13"
```

"Now let's see what we've got."

```
>> SELECT/RETURN ALL↓

PART NUM     : 123769
PRICE        : 39.50
DESCRIPTION  : RD TYR
SUPPLIER     : GOODYEAR
YEAR         : 69
MAKE         : GOODYR
MODEL        : STL 165-13
ON HAND      : 100

PART NUM     : 249321
PRICE        : 41.23
DESCRIPTION  : RD TYR
SUPPLIER     : FIRESTONE
YEAR         : 78
MAKE         : FIREST
MODEL        : FB 165-13
ON HAND      : 90

PART NUM     : 665932
PRICE        : 42.20
DESCRIPTION  : RD TYR
SUPPLIER     : MICHELEIN
YEAR         : 77
MAKE         : MICHEL
MODEL        : STL 165-13
ON HAND      : 126
Path is PARTS BY TYPE:RD TYR:123769
```

Looking up from the terminal, Virginia told the teenager, "We've got Firestone fiber belts for $41.23, but for about the same money you can get the steels -- they're a better tire."

The young man stared at the terminal. "What did you just do?"

Virginia answered modestly, "That's our new computer system -- no more parts catalogs here."

"That's really something. But I didn't think radials would be so expensive...maybe I'll just...well, later..uh..."

Virginia put a stop to his stammering. "Let's take a look at four plys then."

"No, I wouldn't want to put you to the trouble," he quickly countered. "I'm really just pricing them."

"Trouble? Never! You're doing me a favor. I love playing with this thing, besides, you're here already. Let's see what we've got." Virginia returned to her terminal and within a minute rattled off a new series of prices. After a brief discussion, the youth decided to take two four-ply tires.

"I thought I was just price-checking," he said incredulously.

"That's OK," Virginia said with a wink. "I thought I was working."

10:30 A.M.

Virginia was busy at her terminal, reading through the current purchase orders. Her boss, Heather Caucus, strolled by, nodding to the counter help.

"'Morning, Ginny. How's everything going?" Heather asked.

"Hiya Heather. Got a second?" Virginia motioned to the terminal. "A young guy came by this morning looking for radial tires. I ran through all the steel 165-13s but when he wanted something cheaper, I had to go through a whole new series of commands to get the records for fabric belts. Do you think you could make up one of those *automatic sessions* that would automatically list out all the tires?"

"You mean a macro." Heather was obviously pleased by the request. "Give me a seat and I'll try to put one together for you."

Heather sat down in front of the terminal, ready to start typing. "Now pay attention. I'll build a macro that will separately list out all the steel and fabric belted radials, according to size. If it works, I can build another one for the four-plys. Or, you could give it a try. You ought to be able to do this yourself, by now."

In a few minutes, Heather had created two macros. "OK. Let's see how they run. The first one will list the steel belted 165-13s. The second one will list the fabrics."

As Virginia watched, Heather narrated the execution of the first macro file. "First we type in the name of the macro."

```
>> ST_RD_TYR_165_13
```

"The first command positions us in the radial tire subindex."

```
> > PATH PARTS BY TYPE:RD TYR
Path is PARTS BY TYPE:RD TYR
> > DOWN
Path is PARTS BY TYPE:RD TYR:123769
```

"Then we activate the RETAIL qformat and specify the condition that will get us the steel belted radials, size 165-13."

```
> > QFORMAT RETAIL
The current qformat is RETAIL
> > CONDITION MODEL="STL 165-13"
Condition is:     MODEL="STL 165-13"
```

"The SELECT command will pull out all the matching records, keeping our path stationary, so we know where we are when we're done. And we have all the records."

```
> > SELECT/RETURN ALL

PART NUM     : 123769
PRICE        : 39.50
DESCRIPTION  : RD TYR
SUPPLIER     : GOODYEAR
YEAR         : 69
MAKE         : GOODYR
MODEL        : STL 165-13
ON HAND      : 100

PART NUM     : 665932
PRICE        : 42.20
DESCRIPTION  : RD TYR
SUPPLIER     : MICHELEIN
YEAR         : 77
MAKE         : MICHEL
MODEL        : STL 165-13
ON HAND      : 126
Path is PARTS BY TYPE:RD TYR:123769
> >
---- End of Macro file ST_RD_TYR_165_13
```

"That was easy enough. Now for the second macro. It works the same way, except that we've specified a different condition. This time we're looking for fabric belted radials. Notice how I've initially positioned the path. That way, each macro file can run separately."

```
>> FB_RD_TYR_165_13↵
> > PATH PARTS BY TYPE:RD TYR
Path is PARTS BY TYPE:RD TYR
> > DOWN
Path is PARTS BY TYPE:RD TYR:123769
> > QFORMAT RETAIL
The current qformat is RETAIL
> > CONDITION MODEL = "FB 165-13"
Condition is:     MODEL = "FB 165-13"
> > SELECT/RETURN ALL

PART NUM     : 249321
PRICE        : 41.23
DESCRIPTION: RD TYR
SUPPLIER     : FIRESTONE
YEAR         : 78
MAKE         : FIREST
MODEL        : FB 165-13
ON HAND      : 90
Path is PARTS BY TYPE:RD TYR:123769
---- End of Macro file FB_RD_TYR_165_13
>>
```

"You won't even have to position yourself in the subindex, Virginia. Just type in the macro command and the Query will do the rest. Try these out yourself for a while. If you like the way they work, you can build others."

"Thanks Heather. This should really help. And maybe when I've got a chance, I'll make some up for other parts that we're always looking for."

Heather stood up, about to leave. "One more thing," Virginia continued. "What happens when we get this quarter's price and parts changes?"

"Nothing," Heather replied. "Absolutely nothing. We'll just be loading new tapes onto the system -- you won't have to change a thing. Can you imagine -- no more paper cuts from the new manuals."

Virginia just smiled.

11:45 A.M.

Heather Caucus strolled by the accountant's office on the second floor of Caucus Car Parts. Inside the office, she saw Caucus's accountant, Danny Devo, hunched over his Dasher terminal. He looked frustrated.

Heather stuck her head in the door and asked, "What's wrong Danny, isn't the system up?"

Danny swirled in his chair. "Oh, Heather. I'm so glad to see you. I've been trying to do this month's billing on the new system, but it's taking me forever. I spend half my time typing. At this rate, I won't get to the corporate taxes until summer. I'm doing the same work over and over again!"

"Don't you do this billing every month?" Heather asked as she took a seat next to Danny at the terminal.

Danny nodded his head. "Yeah, and that's what I thought the system was supposed to take care of."

"Let's take a look at it, Danny, and see what we can come up with. What procedures do you go through every month for account billing?"

"Well," he began. "I need to get the PO records for all of our manufacturers each month. Oh, and the suppliers, too. That's not so bad. What's really bugging me is that I have to type out all the PO information separately. I need to send it along with the payments. Sorry to say it, Heather, but I hate that typing business -- and the same stuff over and over again -- it just drives me crazy!"

"Calm down, Danny. How would it be if, ah -- instead of your repeatedly typing out all the PO stuff on a separate sheet -- we just sent copies of the records with the PO information that the suppliers need?"

Danny looked puzzled.

"It's really simple, Danny. We can set up a system where you could just select all the purchases of the month from an account like ACCO Manufacturing, display only the record parts you want, and then store all their records in a file. Then all you have to do is read these records, tally up the bill, write a check, and send a copy of the file with the invoice. That way, the line printer can do all the typing, and you can do the accounting, like an accountant is supposed to."

"Can you really do that, Heather? I mean, that's what I had *hoped* we would be able to do, but, well, so far I haven't seen anything like that."

"It's just a matter of taking advantage of what the system has to offer. I'll show you. What information do you usually send along with purchase order information?"

"Let me think a minute. The PO number, of course. And the date, can't forget the date. Then there's the part description, including the year and the model. Let me see, what have I forgotten?" Danny scratched his head. "Oh, yes. The quantity. And the price. Wholesale, not retail. I don't ever let the suppliers see our retail prices."

Heather tried to hide her smile. "I understand. Now, why don't you watch me work the Query. I'll show you how it's done, then you'll know how to use it to develop your own macro sessions later."

With Danny peering over her shoulder, Heather watched her commands echo on the screen. "See what I'm doing now," she said, pointing to the display. "Since I have to edit the qformat file, I'm using the QFORMAT/LINEDIT command."

```
>>QFORMAT/LINEDIT⤶
```

For the next few minutes Heather worked intently at the keyboard, oblivious to her surroundings. "The editing session's finished," she said, cracking her knuckles. "I've created a new qformat that only features the fields you need to send with the PO's. I named it SUPP PO'S, for supplier purchase orders. You should be able to remember that."

```
>>
```

"There's my Query prompt. No errors so we're okay. If I'd made any mistakes in the qformat, the Query would tell me right away. Since I didn't, we can activate the new qformat."

```
>> QFORMAT SUPP PO'S⤶
The current qformat is SUPP PO'S
```

"Then we'll move to the PO record's subindex and read from the database."

```
>> PATH PO⤶
Path is PO
>> DOWN⤶
Path is PO:770945
>> READ⤶

SUPPLIER NAME : ACCO
PO NUM        : 770945
DESCRIPTION   : OIL FT
MONTH,YEAR    : 1179
MODEL         : MONARCH V8
QTY           : 22
SUPP PRICE    : 1.98
```

"See, the new qformat only displays seven fields. And, instead of the full date, it only shows the month and year."

Danny fidgeted with his pencil, stroked his beard, and then finally spoke. "I get it. We now have the record in the form we want. If I need to know how much we ordered from one company, say, ACCO Manufacturing, last month..."

"That's it. You've got it!" Heather excitedly interrupted. "Instead of including the full date, this qformat just features the month and the year. That way, you can access all of ACCO's PO records for a specific month. I set up the system so you can get at the monthly records. And, you can save and copy them, too."

It was Danny's turn to smile. "I see what you're up to. I can isolate all of last month's billing records for ACCO. It's terrific, Heather!"

Heather was already engrossed in her work again, staring at the terminal screen and mumbling to herself. "Next, we'll build a macro file. I have to go back into the CLI and build it there."

```
>> CLI⤶

AOS CLI REV 3.03      3-MAR-80      12:05:51

) X LINEDIT ACCO_MONTHLY_POS.QU⤶
DO YOU WANT THE FILE CREATED? Y⤶

? INSERT⤶
```

"I'll call the file ACCO_MONTHLY_POS; that will be the name of the macro. The commands I put in the macro will position us in the PO subindex, activate this new qformat, SUPP PO'S, and select all ACCO's monthly PO records. And, of course, it will store them all in a separate file.

"Look, Danny." Heather gestured toward the terminal. "These are the commands I want in the macro."

```
QFORMAT SUPP PO'S
PATH PO
DOWN
CONDITION/CLEAR SUPPLIER NAME = "ACCO"
CONDITION/AND MONTH,YEAR = "1179"
SELECT/RETURN/L=ACCO ALL
PATH
CONDITION/CLEAR
QFORMAT/CLEAR
```

"We'll need the QFORMAT command to activate the new qformat -- the one we just created. Then we'll need the PATH command to move to the PO subindex, and a DOWN to get us to the right level for reaching the database. Then we have to declare two conditions: select the PO records for ACCO, but only those that contain the specified month and year. Notice that I included a /CLEAR switch with the first CONDITION command. That way, if any condition was set before using the macro, it will be deactivated. Now, you'll only have to change the month to keep it up to date throughout the year. The next command will actually select the records, and put them in a text file called ACCO. All we'll need then is to just clear up everything. The rest of the commands will do that.

"You can type the text file on the line printer whenever you need it. No more typewriter for you, Danny. The line-printer listings will be faster and more accurate."

" *We* only said *we* didn't like to type, not that *we* were bad at it," Danny quipped.

Heather laughed with him. "Come on, let's try the macro out. If it works, you can use this system for all our major suppliers."

Heather restarted the Query and typed in the macro name. "We have to get out of the CLI and then, to execute the macro, just type its name," she explained.

```
) BYE↵
>> ACCO_MONTHLY_POS↵
```

"Then all those commands we put into the macro start doing their stuff."

```
>> QFORMAT SUPP PO'S
The current qformat is SUPP PO'S
>> PATH PO
Path is PO
>> DOWN
Path is PO:770945
>> CONDITION/CLEAR SUPPLIER NAME = "ACCO"
Condition is: SUPPLIER NAME = "ACCO"
>> CONDITION/AND MONTH,YEAR = "1179"
Condition is: SUPPLIER NAME = "ACCO"
AND           MONTH,YEAR = "1179"
```

"See, the results get sent to a text file with the SELECT command," Heather continued.

```
>> SELECT/RETURN/L=ACCO ALL
        5 Records written into the list file
Path is PO:770945
```

"Watch this, Danny. It's a good idea to reposition and clear the set specifications. That way, you can continue working where you left off without interruption. That's what these last few commands will do."

```
>> PATH
Path is PO:770945
>> CONDITION/CLEAR
No condition is specified
>> QFORMAT/CLEAR
There is no current qformat selected.
---- End of Macro file ACCO_MONTHLY_POS
>>
```

"And there we have it. Let's type out the ACCO file on the printer and take a look at it." She queued the file, and headed for the line printer. "Here it is. What do you think?"

Danny peered over Heather's shoulder and read the listing of ACCO's purchase orders.

```
SUPPLIER NAME  : ACCO
PO NUM         : 770945
DESCRIPTION    : OIL FT
MONTH,YEAR     : 1179
MODEL          : MONARCH V8
QTY            : 22
SUPP PRICE     :  1.98

SUPPLIER NAME  : ACCO
PO NUM         : 780905
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : MONARCH V
QTY            : 12
SUPP PRICE     :  3.12

SUPPLIER NAME  : ACCO
PO NUM         : 780906
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : MUSTANG
QTY            : 10
SUPP PRICE     :  3.05

SUPPLIER NAME  : ACCO
PO NUM         : 780907
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : CAPRI
QTY            : 10
SUPP PRICE     :  2.80

SUPPLIER NAME  : ACCO
PO NUM         : 780945
DESCRIPTION    : OIL FT
MONTH,YEAR     : 1179
MODEL          : FAIRLANE V
QTY            : 22
SUPP PRICE     :  1.98
```

```
SUPPLIER NAME  : ACCO
PO NUM         : 770945
DESCRIPTIONN   : OIL FT
MONTH,YEAR     : 1179
MODEL          : MONARCH V8
QTY            : 22
SUPP PRICE     : 1.98

SUPPLIER NAME  : ACCO
PO NUM         :780905
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : MONARCH V
QTY            : 12
SUPP PRICE     : 3.12

SUPPLIER NAME  : ACCO
PO NUM         : 780906
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : MUSTANG
QTY            : 10

SUPP PRICE     : 3.05

SUPPLIER NAME  : ACCO
PO NUM         : 780907
DESCRIPTION    : GAS FT
MONTH,YEAR     : 1179
MODEL          : CAPRI
QTY            : 10
SUPP PRICE     : 2.80

SUPPLIER NAME  : ACCO
PO NUM         :780945
DESCRIPTION    : OIL FT
MONTH,YEAR     : 1179
MODEL          : FAIRLANE V
QTY            : 22
SUPP PRICE     : 1.98
```

"This is fantastic, Heather. But wouldn't it be better if we could send this information in a more organized form, you know, like a report? It might make it easier for the manufacturers to read."

"You're right Danny. And now that we've set up the SUPPLIER PO'S qformat, we can use that in an rformat to produce a report."

"Isn't that using the Report Writer feature?"

"You've been reading the manual after all! I knew if I left it around, you might look through it."

"Well, I did thumb through it the other day. And, well, this morning, when I was getting so confused..."

"You don't have to explain, Danny. I think it's great you're finally interested in this thing. Look, I'll show you how to set up an rformat. We'll use the /LINEDIT switch with the RFORMAT command to edit the .RFORMS file."

```
>> RFORMAT/LINEDIT↵

? APPEND↵
```

"We'll simply append the new rformat to the end of the file. Let's call it MONTHLY SUPP PO'S. We'll use it to produce a report that we can send along with the payment to each supplier every month. We might as well say so on the comment line."

```
START_REPORT     MONTHLY SUPP PO'S
!        THIS REPORT IS SENT TO EACH SUPPLIER,
!        ALONG WITH THE PAYMENTS, EVERY MONTH
!
```

"Now, let's see. What do we want this report to look like?"

"Well, it would be nice if it was regular width, you know. Then we wouldn't need special paper for the line printer," Danny offered.

"Right, we'll make it 80 columns per line. Then we can also look at the report on the terminal screen without it wrapping. Let's make it 40 lines per page, while we're at it. And, we have to specify that it uses this new qformat. Okay, those are our next three lines."

```
QFORMAT SUPP PO'S
LIN/PG 40
COL/LIN 80
```

"By designating the SUPP PO'S qformat, we're getting all the information from the record that we need. But, we'll also need to calculate a total of all the purchase orders for the month. Let's see, that would be the supplier's price multiplied by the quantity. We have to define that, and give it a picture clause."

```
DEFINE  DOLLAR VALUE     SUPP PRICE * QTY
PICTURE DOLLAR VALUE     $$$$$99V99
```

"We ought to put some headers on this: a title, page number, and date should do it."

```
HEADER 1    C    "CAUCUS CAR PARTS"
HEADER 2    C    "MONTHLY PURCHASER ORDERS"
HEADER 3    60   PAGE
HEADER 4    60   DATE
```

"And we'll need some column headers."

```
HEADER 6      5     "PO NUMBER"
HEADER 7      5     "**********"

HEADER 6      20    "DESCRIPTION"
HEADER 7      20    "***********"

HEADER 6      35    "DATE"
HEADER 7      35    "****"

HEADER 6      45    "MODEL"
HEADER 7      45    "*****"

HEADER 6      60    "QUANTITY"
HEADER 7      60    "********"

HEADER 6      70    "PRICE"
HEADER 7      70    "*****"
```

"Notice that I included asterisks for underlining the headers. It doesn't matter what order we put the HEADER lines in. Now we're ready to define some DETAIL lines. We want the right data items to fall under the right headers."

```
DETAIL 2      5     PO NUM

DETAIL 2      20    DESCRIPTION

DETAIL 2      35    MONTH,YEAR

DETAIL 2      45    MODEL

DETAIL 2      60    QTY

DETAIL 2      70    SUPP PRICE
```

"What about the supplier's name, Heather? Can we include that in the report?" Danny asked.

"Sure, let's put in another DETAIL line for it."

```
DETAIL    1    C    SUPPLIER NAME
```

"And now we can put in our TOTAL lines."

```
TOTAL 3    30    "Total Of Purchase Order"
TOTAL 3    65    TOTAL ( DOLLAR VALUE )
```

"Is that everything?" Danny asked.

"No, we need one more line to indicate the end. And then we'll have everything."

```
END_REPORT
```

"Okay, let's close this file and run the rformat through RWCHECK to see if it's valid. We'll have to get back into the CLI to run RWCHECK."

```
>>CLI
```

AOS CLI REV 3.03 10-JAN-80 12:20:51

```
) X RWCHECK
RFORMS Filename: CAUCUS
```

Would you like to see a list of report format names? [N] N
Report Format Name: MONTHLY SUPP PO'S
START_REPORT MONTHLY SUPP PO'S
! THIS REPORT IS SENT TO EACH SUPPLIER,
! ALONG WITH THE PAYMENTS, EVERY MONTH
!
QFORMAT SUPP PO'S
LIN/PG 40
COL/LIN 80
*DEFINE DOLLAR VALUE SUPP PRICE * QTY*
PICTURE DOLLAR VALUE $$$$$99V.99

```
HEADER 1  C    "CAUCUS CAR PARTS"
HEADER 2  C    "MONTHLY PURCHASE ORDERS"
HEADER 3  60   PAGE
HEADER 4  60   DATE

DETAIL 1  C    SUPPLIER NAME

HEADER 6  5    "PO NUMBER"
HEADER 7  5    "*********"
DETAIL 2  5    PO NUM

HEADER 6  20   "DESCRIPTION"
HEADER 7  20   "***********"
DETAIL 2  20   DESCRIPTION

HEADER 6  35   "DATE"
HEADER 7  35   "****"
DETAIL 2  35   MONTH,YEAR

HEADER 6  45   "MODEL"
HEADER 7  45   "*****"
DETAIL 2  45   MODEL

HEADER 6  60   "QUANTITY"
HEADER 7  60   "********"
DETAIL 2  60   QTY

HEADER 6  70   "PRICE"
HEADER 7  70   "*****"
DETAIL 2  70   SUPP PRICE

TOTAL 3   30   "TOTAL OF PURCHASE ORDER"
TOTAL 3   65   TOTAL ( DOLLAR VALUE)
```

END_REPORT

No errors detected. Want a sample page? [Y] Y↵

Name a file ...any file: [@ OUTPUT] MONTHTEST↵

Do you wish to validate another report format? [N] N↵

)TYPE MONTHTEST↵

"Notice, Danny how I grouped the HEADER and DETAIL lines into clumps. That way, I can look at the rformat, and see what HEADER information goes with what DETAIL information. Now, let's type MONTHTEST and see what the report will look like."

```
                              CAUCUS CAR PARTS
                          MONTHLY PURCHASE ORDERS
                                                                   PAGE   1

                                                                  00/00/00

PO NUMBER     DESCRIPTION    DATE       MODEL           QUANTITY    PRICE

                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX
                              XXXXXXXXXX
XXXXXX        XXXXXX         XXXX       XXXXXXXXXX      XX          XXXXXX

                                        TOTAL OF PURCHASE ORDER     $9999.99
```

"Heather, that doesn't look right," Danny pointed to the screen. "What's that line of Xs in between all the other lines?"

"Oh dear. That's supposed to be the supplier's name. I forgot something. I put that in as a DETAIL line. DETAIL information isn't printed until after all the HEADER information is printed. We'll have to change that. I'll put the supplier name data item in as another HEADER line instead of as a DETAIL line. Then it will appear at the top of the page before the columns start. How does that sound?"

"Sounds good. But before you do that, explain to me how we're going to use this report. I mean, do I have to use it for each supplier?" Danny asked.

"Well, you could set up Condition Statements each time, specifying which supplier you want and what month. Or, we could put in a SORT line so the records will be sorted by the supplier's name. Let me think about this a minute." Heather stopped to consider the possibilities.

"I've got it. We'll sort by supplier name, and put a page break in. Everytime the Report Writer hits a new supplier name, it will start printing on a new page. That means we'll want subtotals at the end of each page instead of one large total at the end of the report. Then, you'll be able to run this whole thing one time a month, get a report out for each supplier, complete with the total of their purchase orders."

"This thing gets better and better. The Query does more and more work and I do less and less. I'll still have a job when we're through with this, won't I?"

"Of course, Danny," Heather laughed. "It will just be simpler, and maybe, even more enjoyable. Let me first change this rformat, and we'll see how it works."

Heather invoked a text editor and made some changes to the CAUCUS.RFORMS file. When she was through, she ran RWCHECK again and asked for a sample page.

"Okay, Danny, take a look at this. Here's the new rformat."

```
START_REPORT      MONTHLY SUPP PO'S
!                 THIS REPORT IS SENT TO EACH SUPPLIER,
!                 ALONG WITH THE PAYMENTS, EVERY MONTH
!
QFORMAT           SUPP PO'S
LIN/PG            40
COL/LIN           80
DEFINE            DOLLAR VALUE                    SUPP PRICE * QTY
PICTURE           DOLLAR VALUE                    $$$$$99V.99

HEADER    1       C                               "CAUCUS CAR PARTS"
HEADER    2       C                               "MONTHLY PURCHASE ORDERS"
HEADER    3       60                              PAGE
HEADER    4       60                              DATE

SORT              SUPPLIER NAME

HEADER    4       C                               SUPPLIER NAME
BREAK             SUPPLIER NAME                   PAGE_EJECT
BREAK             SUPPLIER NAME                   PRE_BREAK_SPACE
BREAK             SUPPLIER NAME                   30    "TOTAL OF PURCHASE ORDER:"
BREAK             SUPPLIER NAME                   65    TOTAL (DOLLAR VALUE)

HEADER    6       5                               "PO NUMBER"
HEADER    7       5                               "*********"
DETAIL    2       5                               PO NUM

HEADER    6       20                              "DESCRIPTION"
HEADER    7       20                              "***********"
DETAIL    2       20                              DESCRIPTION

HEADER    6       35                              "DATE"
HEADER    7       35                              "****"
DETAIL    2       35                              MONTH,YEAR

HEADER    6       45                              "MODEL"
HEADER    7       45                              "*****"
DETAIL    2       45                              MODEL

HEADER    6       60                              "QUANTITY"
HEADER    7       60                              "********"
DETAIL    2       60                              QTY

HEADER    6       70                              "PRICE"
HEADER    7       70                              "*****"
DETAIL    2       70                              SUPP PRICE

END_REPORT
```

"Notice the BREAK lines I added. The PRE_BREAK_SPACE clause tells the Report Writer to skip a line after printing the DETAIL information. The PAGE_EJECT clause tells it to start a new page when the supplier name changes, and the other two BREAK lines indicate the summary information we want included on the subtotal lines.

"And here's the sample page."

CAUCUS CAR PARTS
MONTHLY PURCHASE ORDERS

PAGE 1

XXXXXXXXXX *00/00/00*

PO NUMBER	*DESCRIPTION*	*DATE*	*MODEL*	*QUANTITY*	*PRICE*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
XXXXXX	*XXXXXX*	*XXXX*	*XXXXXXXXXX*	*XX*	*XXXXXX*
			TOTAL OF PURCHASE ORDER:		*$9999.99*

"Now, let's try it out with real data."

Heather started a new Query session.

) X QUERY/Q=MONTH_REPORT.QU CAUCUS⤶
INFOS Query / Report Writer .... At Your Service

"What's the /Q for?" Danny asked.

"I'm using that switch to create a macro. All the commands I type during this Query session will be saved in the file MONTH_REPORT.QU. Then, when we want to repeat this whole session, we simply use the name of the macro as a command. And, if we want to make changes to the macro, we simply edit the MONTH_REPORT.QU file." Heather explained as she continued typing. She positioned the PATH to the PO subindex and asked the Query to READ all the records and display them in the form of a report.

```
>> PATH PO↵
Path is PO
>> DOWN↵
Path is PO:770945
>> RFORMAT/NAMES↵
The following rformats have been defined :
        STOCK VALUE
        OLDIES
        MONTHLY SUPP PO'S
>> RFORMAT MONTHLY SUPP PO'S↵
The current qformat is SUPP PO'S
The current rformat is MONTHLY SUPP PO'S
>>READ/RFORMAT/RETURN/L=MON_REPORT ALL↵
        55 pages written into the list file.
Path is PO:770945
>> RF/CLEAR↵
There is no current rformat selected
>> BYE↵
```

"Fifty-five pages! That's every purchase order we've made in the past two years!"

"Of course, Danny. That's because I used the READ command. You will be using a SELECT command when you use this rformat. You'll set a CONDITION, and then SELECT all the records that match that CONDITION. Here. I'll show you. You're working on January's PO's, right?"

Danny nodded his head.

"All right. I'll set a CONDITION that will get us all those purchase order records." Heather began typing. "Let's check our position and activate the rformat first."

```
>> PATH↵
Path is PO:770945
>> RFORMAT MONTHLY SUPP PO'S↵
The current qformat is SUPP PO'S
The current rformat is MONTHLY SUPP PO'S
```

"Then we set the CONDITION, and use the SELECT command. I'll count the number of records first, using the /COUNT switch."

```
>> CONDITION MONTH,YEAR = "0180"↵
Condition is: MONTH,YEAR = "0180"
>> SELECT/COUNT/RETURN ALL↵
        19 Records Selected
```

"See. There are 19 PO records for January. Do you want to see them?"

"Send them to a file so we can have a line-printer listing."

"All right. Notice that I'm using the /RFORMAT switch to get them in a report."

```
>> SELECT/RETURN/L=JAN_PO/RF ALL↵
        9 pages written into the list file
Path is PO:770945
```

"Now, I can build you a macro that would do all this. I'll edit the MONTH_REPORT.QU file to contain just the commands you'll need to SELECT the monthly purchase order information. Each time you do the monthly purchase orders, you'll have to change the CONDITION to indicate the month you're working on, and the name of the file the records are sent to. Then you'll simply run the macro and have the reports all ready to send to the suppliers. I'll call this other macro MONTH_SUPPLIER.QU."

Heather busied herself at the terminal for a few minutes. "Here, these are the commands in the macro."

```
RF MONTHLY SUPP PO'S
PATH PO
DOWN
CONDITION/CLEAR MONTH,YEAR = "0180"
SEL/RF/RETURN/L=JAN_PO ALL
CON/CLEAR
```

"Now you have three macros. One if you need the records displayed in a qformat, one to produce a report, and one to READ all the PO records into a report."

"Thanks, Heather. I'm impressed. The Query and these Report Writer features are great. Now, if you don't mind, I want to play with them myself."

"Be my guest. The terminal is all yours."

Heather stood up and Danny immediately took her seat. "Don't overdo it, Danny. Save something for tomorrow."

"Don't worry. I just want to practice what I've learned."

Heather smiled, and walked out.

3:00 P.M.

Dave Harris sat in his office pouring over the latest inventory figures. He had a troubled look on his face.

Suddenly, he whirled his chair around so that he was facing his terminal. He started typing away at the keyboard. After a few minutes, he ripped some pages off the line printer, scanned them quickly, and walked briskly out of his office.

3:30 P.M.

Dave Harris walked into Heather's office and stopped when he saw her at the console.

"Oh, hi, I was just checking out our overhead Dave, been very busy you know, lots of work," she mumbled, finishing up her keyboard sequence and logging off. She turned and gave Dave her full attention.

"I was looking over the latest inventory figures and we've got trouble. Take a look at these reports." Dave waved the papers in his hand. "And, speaking of trouble, I just passed Danny's office. Did you see what he's doing with the Query? Do you think it's okay to let him play around with it? It's not a toy!"

Heather smiled. "Oh Dave! I just showed him some purchaser order routines and helped him set up some macros and an rformat. He's just practicing, and he's finally getting the hang of it. Besides, you know Query is read only."

"Well, I guess that's all right," Dave mumbled. "But look at these reports. I had a meeting with my father this morning and he wanted to know if we can use the computer to answer some questions we've been asking ourselves lately. Your mother was there, too. She seems to think this system is the living end. But I guess, like daughter like mother."

"Dave! You're so skeptical."

"I still don't see what's so bad about writing application programs. I did it for years, and we always had the information we needed."

"But, Dave. Not everyone who works here is a programmer like you. Why should you be the only one who has access to the information?"

"Never mind that now. I'll argue with you some other time. Right now we've got trouble."

"You keep saying that. Would you please explain yourself?"

"Well, Dad suggested we start cleaning out our stock. And he thinks the Query might help. So I produced a few reports, and do you realize how much old stuff we've got hanging around? Not only do we still carry items for cars over 10 years old, we even have orders in for more! Look at these figures." Dave put the papers he'd been clutching on Heather's desk. "Do we really need to stock five horns for '52 Fords? And have four more on order? And look at this one! Twelve distributer caps on order for '66 Fords! It's outrageous!"

"Calm down. We do have surplus stock choking the stockrooms. Old stuff. Parts that don't move. I know we do have a couple of regular customers that own '49 Edsels, but we don't need so many gas caps on hand. Your father's right. We ought to clean up this place."

"That's what your mother said. She also pointed out that we're paying too much for foreign parts. She suggested we start stocking some American-made replacements."

"That's a good idea. We need to get rid of outdated parts, right. What else do we need to get rid of?" Heather warmed up to the problem.

"I suppose we could start with convertible parts--we haven't had much of a demand for them since Detroit stopped making ragtops. And they take up a lot of shelf space," Dave said, hesitantly.

"Especially since you insisted on ordering so many that time..."

"Okay, okay. Don't rub it in. We all make mistakes. Let me show you how I got these reports."

Dave sat down at the terminal and logged on to the Query. "First I checked to see how many records we have for items over 10 years old."

```
>> PATH PARTS↵
Path is PARTS
>> DOWN↵
Path is PARTS:056793
>> RFORMAT STOCK VALUE↵
The current qformat is PARTS
The current rformat is STOCK VALUE
>> CONDITION YEAR < 69↵
Condition is:          YEAR < 69
>> SEL/COUNT/RETURN ALL↵
        153 Records Selected
Path is PARTS:056793
```

Heather watched over Dave's shoulder. "Why did you activate the STOCK VALUE rformat first?" She asked.

"It uses the PARTS qformat, and look at the fields it displays." Dave typed in another command to display the qformat for Heather.

```
>> QF/DISP↵
PART NUM            1    6    ASCII
DESCRIPTION         7    12   ASCII
YEAR                13   14   ASCII
MAKE                15   20   ASCII
MODEL               21   30   ASCII
ON HAND             31   33   ASCII
IN WAREHOUSE        34   36   ASCII
REORDER             37   39   ASCII
SUPPLIER            40   49   ASCII
SUP ADDRESS         50   59   ASCII
RETAIL PRICE        60   65   ASCII
WHOLESALE PRICE     66   71   ASCII
```

"See. Using this qformat, we can look for the records of the old parts, plus the ones that are on reorder. In fact, it turns out that all the old parts are being reordered. Look at this." Dave added another CONDITION, and used the SELECT command with the /COUNT switch.

```
>> CONDITION/AND REORDER > 0↵
Condition is: YEAR < 69
        AND   REORDER > 0
>> SEL/COUNT/RETURN ALL↵
        12 Records Selected
```

"Twelve records selected this time, too. Do you want to see them?" Dave asked.

"That's okay. I can look at your line-printer listing. Besides, I get the picture. In fact..."

"Then, I SELECTED all these records using the /RFORMAT switch." Dave interrupted. "Let me show you."

"That's okay, Dave. I see what you're getting at. But what we really need to see is the value of the stock broken down by year and how much revenue can be generated by returning parts. The OLDIES rformat does just that."

"OLDIES?" Dave asked.

"Are you familiar with that rformat? First change the rformat which will clear the CONDITION. Better yet, let me try it." Heather literally pushed Dave out of the chair and started typing.

```
>> CONDITION/CLEAR↵
No condition is specified
>> RF OLDIES↵
The current qformat is PARTS
The current rformat is OLDIES
```

"I'll display OLDIES for you."

```
>> RF/DISP↵
```

```
START_REPORT    OLDIES
!          A REPORT FOR MANAGEMENT TELLING HOW MUCH REVENUE
!          CAN BE GENERATED BY RETURNING OLD PARTS
HEADER  1  C   "CAUCUS CAR PARTS"
HEADER  2  C   "Return Value of Old Inventory"
HEADER  3  60  PAGE
HEADER  4  60  DATE
LIN/PG     30
COL/LIN    80
DEFINE     DOLLAR VALUE   ON HAND * WHOLESALE PRICE
DEFINE     RETURN VALUE   90 % DOLLAR VALUE
PICTURE    RETURN VALUE   $$$$$99V.99
SORT       YEAR
BREAK      YEAR  PRE_BREAK_SPACE
BREAK      YEAR  PRE_BREAK_SPACE
BREAK      YEAR  PRE_BREAK_SPACE
           YEAR  21   "Return Value of"
BREAK      YEAR  38   YEAR
BREAK      YEAR  41   "Stock = "
BREAK      YEAR  52   TOTAL ( RETURN VALUE )
END_REPORT
```

"I worked out this rformat when we first got the Report Writer." Heather explained. "Mother was bugging me then about overstocking. I just never got a chance to do anything about it."

"Well, let's see what it gives us."

"All right. I'll set a new CONDITION and SELECT all the matching records using the /RFORMAT switch."

```
>> PATH↵
Path is PARTS:056793
>> CONDITION YEAR < 70↵
Condition is: YEAR < 70
>> SEL/RETURN/RF ALL↵
        171 records will be sorted
```

"It's sorting 171 records," Dave pointed to the screen. "Is that why we aren't getting any response?"

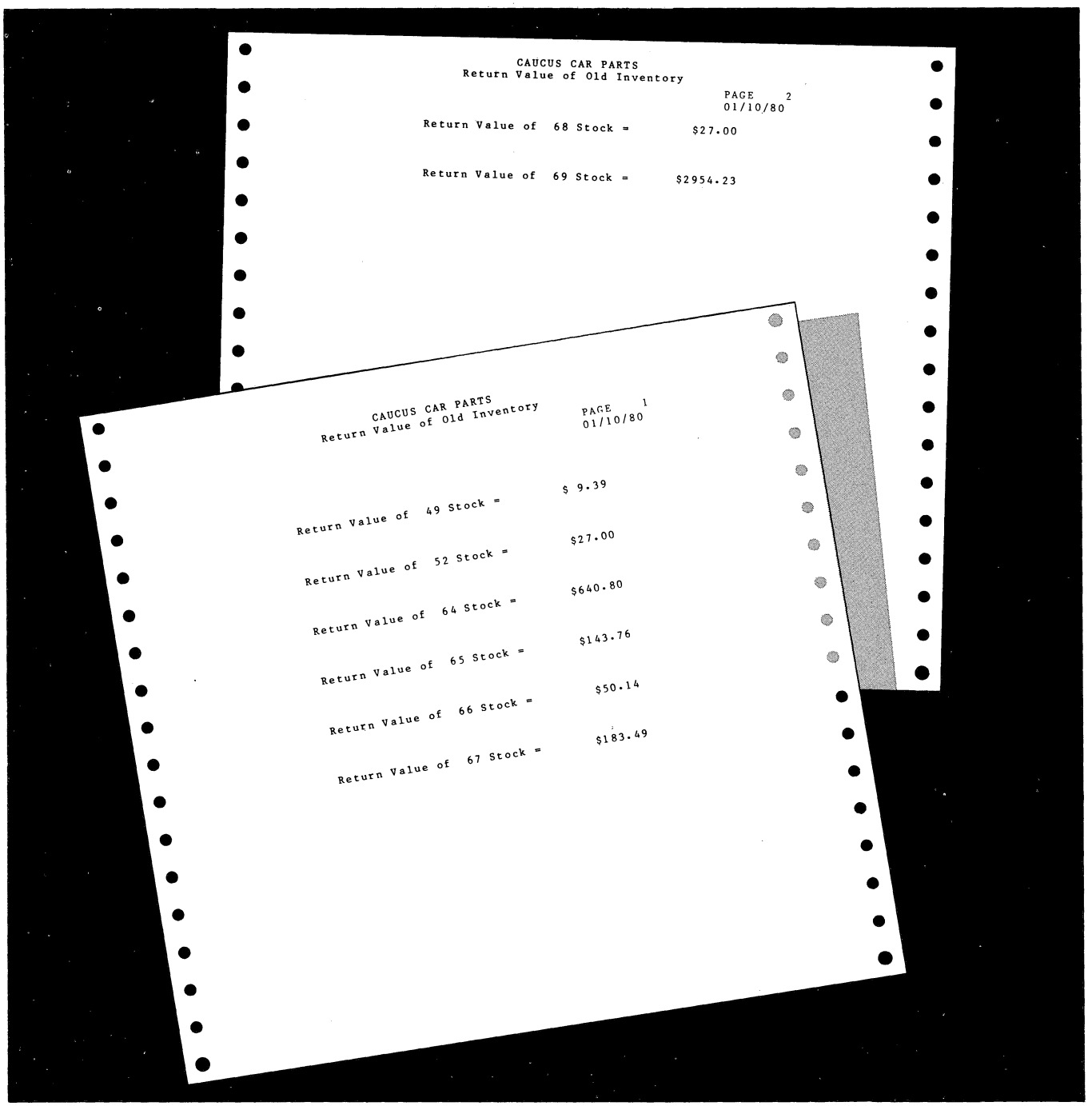
"Yes. We won't get a prompt until all the records are sorted. We want the value of the 1969 stock, too. You specified all records for the years less than 1969. So it's even more parts than you thought. Look, here's the report now."

"Wow! Look what we can save on the 1969 stock alone!" Dave exclaimed. "Let's send all the records to Receiving and have the shippers send the parts back to the manufacturers."

"Which records? Are we still talking about the convertible parts? Let's take a look at them too." Heather repositioned the path, changed the qformat and CONDITION, and SELECTed all the matching records.

```
>> PATH PARTS↵
Path is PARTS
>> DOWN↵
Path is PARTS:056793
>> QFORMAT PARTS↵
The current qformat is PARTS
>> COND MODEL = "CNV"↵
Condition is: MODEL = "CNV"
>> SEL/RETURN ALL↵

PART NUM          : 057211
DESCRIPTION       : BRA PD
YEAR              : 69
MAKE              : VOLVO
MODEL             : CNV 18005
ON HAND           : 4
IN WAREHOUSE      : 3
REORDER           : 5
SUPPLIER          : VOLVO AM
SUP ADDRESS       : RALEIGH NC
RETAIL PRICE      : 16.23
WHOLESALE PRICE : 13.12

PART NUM          : 057217
DESCRIPTION       : BRA PD
YEAR              : 70
MAKE              : VOLVO
MODEL             : CNV 18005
ON HAND           : 2
IN WAREHOUSE      : 5
REORDER           : 5
SUPPLIER          : VOLVO AM
SUP ADDRESS       : RALEIGH NC
RETAIL PRICE      : 16.78
WHOLESALE PRICE : 13.26

PART NUM          : 057219
DESCRIPTION       : BRA PD
YEAR              : 72
MAKE              : VOLVO
MODEL             : CNV 18005
ON HAND           : 4
IN WAREHOUSE      : 7
REORDER           : 5
SUPPLIER          : VOLVO AM
SUP ADDRESS       : RALEIGH NC
RETAIL PRICE      : 16.57
WHOLESALE PRICE : 13.21

PART NUM          : 222542
DESCRIPTION       : TRANS
YEAR              : 74
MAKE              : TRMP
MODEL             : CNV SPTFIR
ON HAND           : 1
IN WAREHOUSE      : 1
REORDER           : 2
SUPPLIER          : BR LEYLAND
SUP ADDRESS       : LONDON GB
RETAIL PRICE      : 123.40
WHOLESALE PRICE : 100.62

PART NUM          : 333229
DESCRIPTION       : GS TNK
YEAR              : 76
MAKE              : FORD
MODEL             : CNV IMP V8
ON HAND           : 2
IN WAREHOUSE      : 1
REORDER           : 3
SUPPLIER          : FO MO CO
SUP ADDRESS       : DETROIT MI
RETAIL PRICE      : 23.40
WHOLESALE PRICE : 18.10
```

"Hold it!" Dave called out. "These parts are for newer cars!"



SD-02019

Heather quickly used the interrupt command to stop the Query's record display.

↑C↑A
CONSOLE INTERRUPT

"We don't want to get rid of these parts." Dave said. "Not for 1976 cars or VW's."

"Do you want me to add another CONDITION?" Heather asked. "So we only get the old convertible parts."

"Yes. And we don't want any VW parts."

Heather began typing again.

```
>> CON MODEL = "CNV"↓
Condition is: MODEL = "CNV"
>> CON/AND YEAR < 70↓
Condition is: MODEL = "CNV"
        AND   YEAR < 70
>> CON/AND MAKE <> "VW"↓
Condition is: MODEL = "CNV"
        AND   YEAR < 70
        AND   MAKE < > "VW"
```

"We're looking for all the records for convertibles, older than 1970, and not VW's. Right?" Heather checked with Dave.

"That's right. And why don't you send them straight to a file. Then we can have someone in the stockroom start sending them back to the manufacturers."

"Good idea."

```
>>UP↓
Path is PARTS
>> DOWN↓
Path is PARTS:056793
>> SEL/RETURN/L=OLD_CONV ALL↓
        24 Records written into the list file
Path is PARTS:056793
```

"Only 24 records," Dave commented.

"It's a start. Besides, with all the other old parts, it will at least make a dent. We'll free up enough space that we can take on some additional lines. I'll send those records to a file, too."

```
>> CONDITION/CLEAR↓
No condition is specified
>> CONDITION YEAR < 70↓
Condition is: YEAR < 70
>> SELECT/L=OLD_PARTS ALL
        47 records written into the list file
```

"Well, that takes care of that." Heather stood up, as if to leave.

"Hold on, Heather. Now there's that business of foreign parts. We've been taking a killing on the devaluation of the dollar -- German and Japanese parts are pricing us out of the market. Your mother suggested we try to stock more American parts, wherever possible. Do you think you could estimate how much a ten percent devaluation would cost us? Whether American parts would be cheaper or not?"

Heather started putting on her jacket. "No problem Dave. You can do it yourself. You need some more practice on the Query. Good luck. I've got to run."

She waved as she walked out of the door, leaving Dave standing by the terminal. "But Heather!"

Dave stared at the closed door for a few seconds. "How do you like that," he mumbled to himself. "She doesn't think I can do it. I'll show her!"

He sat down at the terminal and began churning the Query problems around in his head.

"We need all the Japanese and German parts. If I have them, I can match the part numbers up with American suppliers, list out the two groups -- in separate files -- and then compare the prices."

Dave mumbled to himself as he started typing. "Let's see what the different qformats are."

```
>> QF/NAMES↓
The following qformats have been defined on this file :
        RETAIL
        PARTS
        OPEN PO'S
        TYR PO LOCATIONS
        SUPP PO'S
```

"Hmm. I don't think any of these will do the trick. I better create a new one that will easily tell me the supplier's country. I'll call it SUPP LOCATIONS."

```
>> QFORMAT/LINEDIT↓
```

A few minutes of typing, and Dave sat back in his chair. "That should do it," he mused.

```
>> QF SUPP LOCATIONS↓
The current qformat is SUPP LOCATIONS
```

"Let's check it out, just to be sure."

```
>> QF/DIS↓
PART NUM      1    6    ASCII
DESCRIPTION   7    12   ASCII
YEAR          13   14   ASCII
MAKE,MODEL    15   30   ASCII
SUPPLIER      40   49   ASCII
COUNTRY       58   59   ASCII
```

"First, I'll specify the CONDITIONS to pull out all the Japanese and West German parts. I better reposition the PATH, too."

```
>> COND COUNTRY = "JP"↓
Condition is: COUNTRY = "JP"
>> CON/OR COUNTRY = "WG"↓
Condition is: COUNTRY = "JP"
        OR     COUNTRY = "WG"
>> PATH PARTS↓
Path is PARTS
>> DOWN↓
Path is PARTS:056793
```

"Now I can send them to a file."

```
>> SEL/RETURN/L=FOREIGN_PARTS ALL↓
        9 Records Selected
Path is PARTS:056793
```

"Now I should check what parts we stock for foreign cars. I'll change the qformat first. I think PARTS should do the trick for me, but I better check it out."

```
>> QFOR PARTS↓
The current qformat is PARTS
>> QF/DIS↓
PART NUM          1    6    ASCII
DESCRIPTION       7    12   ASCII
YEAR              13   14   ASCII
MAKE              15   20   ASCII
MODEL             21   30   ASCII
ON HAND           31   33   ASCII
IN WAREHOUSE      34   36   ASCII
REORDER           37   39   ASCII
SUPPLIER          40   49   ASCII
SUP ADDRESS       50   59   ASCII
RETAIL PRICE      60   65   ASCII
WHOLESALE PRICE   60   71   ASCII
```

"This is the one. All I have to do is set a lot of CONDITIONS for MAKE."

```
>> CON MAKE = "TOYOTA"↓
Condition is: MAKE = "TOYOTA"
>> CON MAKE = "DATSUN"↓
You must specify either /AND or /OR
```

"Hmm. Do I want /AND or /OR? I better ask for HELP."

```
>> HELP CONDITION↓
```

CONDITION

Sets or displays a selection criterion.

Syntax *CONDITION[switches] [<item1> <condition> <item2>]*

Switches	*/AND*	*Logically AND this relation to the current condition.*
	/OR	*Logically OR this relation to the current condition.*
	/CLEAR	*Clears the current condition.*
<items>	*may be literal strings, numbers, or datanames in a qformat.*	
<conditions>	*may be: "=", "<", "<=", "=<", "<>", "=>", ">=", or ">"*	

Use *CONDITION sets or displays selection criteria to be used by the SELECT command. With no argument, CONDITION displays the current condition. The selection criteria depends upon the current qformat which identifies data names and data types. When you establish a new qformat, QUERY clears the previous condition. The argument you use with the CONDITION command creates a simple relation between data names and/or constants. Once have established one relation, you can append others to it using the /AND or /OR switch. If you include both /AND and conditions, the /OR statements should usually be first because the Query evaluates the expression in a left to right order.*

For an example of the CONDITION command, type:
>> HELP EX_CONDITION

"I need to use the /OR switch."

```
>> CON/OR MAKE = "DATSUN"↓
Condition is: MAKE = "TOYOTA"
        OR      MAKE = "DATSUN"
```

"That one worked. Let's add another CONDITION."

```
>> CON/OR MAKE = 'SUBARU'↓
Not a legal variable -- PHRASE:         2
```

"Now what's wrong? Oh. I used single quotes instead of double quotes. I'll try again."

```
>> CON/OR MAKE = "SUBARU"↓
Condition is: MAKE = "TOYOTA"
        OR      MAKE = "DATSUN"
        OR      MAKE = "SUBARU"
>> CON/OR MAKE = "BMW"↓
Condition is: MAKE = "TOYOTA"
        OR      MAKE = "DATSUN"
        OR      MAKE = "SUBARU"
        OR      MAKE = "BMW"
>> CON/OR MAKE = "MERCED↓
Not a legal variable -- PHRASE:         2
```

"Oops! Forgot the closing quote. I must be getting tired. Well, I only need a few more conditions." Dave kept typing, adding CONDITION after CONDITION, until he exceeded the maximum allowed clauses.

```
>> CON/OR MAKE = "FIAT"↓
Condition is: MAKE = "TOYOTA"
        OR      MAKE = "DATSUN"
        OR      MAKE = "SUBARU"
        OR      MAKE = "BMW"
        OR      MAKE = "MERCED"
        OR      MAKE = "RENAUL"
        OR      MAKE = "PORSCHE"
        OR      MAKE = "MG"
        OR      MAKE = "VW"
        OR      MAKE = "AUDI"
        OR      MAKE = "TRMP"
        OR      MAKE = "PEUGEOT"
        OR      MAKE = "VOLVO"
        OR      MAKE = "SAAB"
        OR      MAKE = "FIAT"
>> CON/OR MAKE = "HONDA"↓
Maximum number of Condition clauses exceeded
```

"That does it! It's almost 5 o'clock. I'll SELECT these records, put them in a file, and compare it to the FOREIGN_PARTS file. At least it's a start. I'll be able to find out most of the foreign parts we stock by comparing the two files, and which of those we can get from American suppliers. I'm sure that's more than Heather expected me to do."

```
>> PA↓
Path is PARTS:056793
>> SELECT/RETURN/L=ALL_FOREIGN_PTS ALL↓
194 Records written into the list file
Path is PARTS:056793
```

"Good grief!" Dave looked up from the terminal and stared at the clock. "I've been talking to myself long enough."

Dave logged off and looked around the office. "I hate to admit it, but this Query thing is all right. It's even fun! And to think that Ginny and Danny now have access to all the information they need to do their jobs! Plus, no more painful weeks of writing and rewriting application programs. Quite an invention, this INFOS Query system." Dave quickly looked over his shoulder. "Better not let Heather know that yet. She'll never let me live it down!"

End of Chapter

Appendix A
Query Function Limits

Function	Limit	Explanation
Maximum key length you can specify	255 bytes	This is the INFOS system key length limitation.
Maximum level of subindexing	8	This is a Query limit.
Maximum record length you can specify	2048 bytes	This is shorter than the INFOS limit.
Maximum depth to which you can embed macros	5	The Query will not accept embedded macros beyond this limit.
Maximum number of conditions you can specify in a CONDITION statement	15	This limit includes both /ANDs and /ORs.
Maximum number of QFORMATS you can create in a .QFORMS file	10	You can specify 10 qformats in each .QFORMS file. You may have several .QFORMS files.
Maximum number of RFORMATS you can create in a .RFORMS file	10	You can specify 10 rformats in each .RFORMS file. You may have several .RFORMS files.
Maximum number of field descriptor lines in a qformat	30	The Query will not accept more than 30 field descriptor lines in an individual qformat.
Maximum number of BREAK data_item pairs per rformat	5	You can define up to 5 BREAK actions for each data item in an rformat.
Maximum number of DEFINE variables per rformat	10	You can define 10 temporary variables in an rformat.
Maximum number of HEADER output lines per rformat	10	You can define up to 10 lines of HEADER information for one report.
Maximum number of DETAIL output lines per rformat	5	You can define up to 5 lines of DETAIL information for one report.

Note: All limits are subject to change. See the current INFOS Release Notice for changes.

Function	Limit	Explanation
Maximum number of TOTAL output lines per rformat	10	You can define up to 10 lines of TOTAL information for one report.
Minimum lines per page for a printed report	30	An rformat must allow at least 30 lines for a printed report.
Default value for COL/LIN command line in an rformat	80	An rformat automatically allows 80 characters per line, unless otherwise specified.
Default value for LIN/PG command line in an rformat	60	An rformat automatically allows 60 lines per page, unless otherwise specified.
Maximum number of SORT lines per rformat.	5	You can define up to 5 SORT lines in an rformat.

Note: All limits are subject to change. See the current INFOS Release Notice for changes.

End of Appendix

Appendix B
Error Messages

This Appendix includes explanations of the error messages you might receive while working with the Query and Report Writer. We also suggest ways to correct these errors.

The error messages are arranged as follows:

- General Query
- Query Movement
- Query Selection and Display
- Miscellaneous Query
- Qformat syntax
- Report Writer runtime
- Rformat syntax

General Query Error Messages

You'll receive the following errors after issuing a variety of erroneous Query commands.

Command abbreviation not unique

You have not given the Query enough letters; it can't tell which command you're trying to abbreviate. Use a longer abbreviation.

Command requires argument

You didn't use a required argument; if you don't know which arguments are allowed, type HELP or check Chapter 6 of this manual.

Command does not take an argument

You've used an argument with a command that doesn't need one. Check the syntax of the command by typing HELP, or look in Chapter 6 of this manual.

Argument must be an integer between 1 and 32767

This command takes an integer between 1 and 32767, or ALL as an argument. If you have not specified a number within the proper range, try again, or check the command's syntax.

Switch(es) not allowed

You've tried to use a switch on a command that doesn't take one. Check the command syntax.

---x--- is not a valid switch for this command

You've specified an invalid switch for this command; check the syntax.

Ambiguous switch ---x---

The Query isn't sure what switch you want; your abbreviation is not unique, try again.

Unable to open list file.

Query was unable to open the file specified with the "/L=" switch. Try using a different filename.

Unable to open macro file

Query found the specified macro file, but AOS won't open it. Check the ACL's on the file.

Attempt to exceed macro recursion depth

You've tried to embed more than five macros; Query won't allow that.

Huh ? - That's not a Query command or a Query macro

Whatever you typed has confused the Query; it's not a recognizable command. Try again.

Unexpected EOF on primary INPUT file !!

You tried to run Query in batch using the /B=filename switch. Check the input file; there should be a BYE command at the end of it.

Query Movement Command Errors

You'll receive the following error messages after issuing one of the Query movement commands. The error messages are arranged by command. If an error occurs, your path remains unchanged.

KEY Command

Keyed search was unsucessful

The key you specified doesn't exist, or at least the Query can't find it at this level. Either reposition your Path and try again, or try a different key.

PATH Command

Path is above the index

You're positioned too high in the index structure. Do a DOWN command before trying the PATH command again.

Path has too many levels

You've used an argument that has more levels than the maximum allowed in this file.

UP, DOWN, NEXT and PRIOR Commands

Argument must be an integer between 1 and 32767

If you try to indicate a number of keys you want to move, in any direction, you must either use an integer between 1 and 32767 or ALL as an argument.

Sorry, you are already at the top

You are at the highest level index - you can't go UP anymore. Try a DOWN command.

Path has too many levels

You are at the lowest level index - you can't go DOWN anymore. Try an UP command.

Alas - no more entries in this subindex

If you receive this message after issuing a PRIOR command, you're at the front end of the subindex and can't do a PRIOR. If you received this message after issuing a NEXT command, you're at the back end of the subindex and can't do a NEXT.

Path is above the index

You're positioned too high in the index structure. Try a DOWN command.

Oops ! No subindex under the current path

You can't move DOWN unless there is a subindex defined under the current key.

Unexpected INFOS error < error message >

In most cases, you'll receive this message if something is wrong with your INFOS file. Check with someone at your installation who knows more about INFOS.

Query Selection and Display Command Errors

You'll receive the following error messages after issuing one of the Query selection and display commands. The error messages are arranged by command in alphabetical order.

CONDITION Command

Maximum number of Condition clauses exceeded

You receive this message when you try to specify more than 15 Condition clauses. You must either clear the Condition and start again, or limit your request to 15 Conditions.

There is no current qformat selected

If you try to specify a Condition before activating a qformat, you'll receive this message. You must activate a qformat or rformat before specifying a Condition.

This is the first clause - what do you mean /AND or /OR

The first Condition clause specifed does not take an /AND or an /OR switch. Try again.

You must specify either /AND or /OR
You can't specify both /AND or /OR

You'll get either of these message when you've already defined Condition clauses. You must use either /AND or /OR with each additional Condition clause, or /CLEAR and start again.

No legal relational operator found

You must include one of the following relational operators in a Condition clause: = < , < = , < > , < , = , > , > = , = > .

Not a legal variable -- PHRASE < 1 or 2 >

A Condition clause must contain two phrases, separated by a relational operator; these phrases can be a number, a dataname, or a string.

READ and SELECT Commands

Path is above the index

You are positioned too high in the index structure. Do a DOWN command before READing or SELECTing any records.

No condition is specified

You cannot SELECT without first specifing a condition.

No current rformat selected

You tried to use the /RFORMAT switch before activating an rformat.

Argument must be an integer between 1 and 32767

You can ask the Query to READ or SELECT up to 32767 records. The only other allowable argument is ALL.

Key has no database record

You've asked the Query to READ or SELECT a record but your path is not DOWN far enough, the key you specified does not exist, or the key does not have a record associated with it. With a READ command, this key will be ignored, but counted. With a SELECT command, Query will not select the record.

Unexpected INFOS error
< error message >

In most cases, you'll receive this message if something is wrong with your INFOS file. Check with someone at your installation who knows more about INFOS.

COUNT & RFORMAT are incompatible switches: RFORMAT is ignored

You've specified incompatible switches. Query ignores the /RFORMAT switch, but performs the /COUNT.

RFORMAT & SCREEN are incompatible switches: RFORMAT is ignored

You've specified incompatible switches. Query ignores the /RFORMAT switch, but performs the /SCREEN.

SCREEN & L are incompatible switches: L is ignored

You've specified incompatible switches. Query ignores the /L but performs the /SCREEN.

SCREEN only works on Dasher display terminals
SCREEN implies I/O at the terminal. Switch ignored.

You'll get either of these errors if you used the /SCREEN switch and you're not working at a DASHER ™ display terminal.

Too many Columns per Line for SCREEN format

You've tried to use the /SCREEN switch after activating an rformat defined with more than 80 Columns Per Line. You can't use the /SCREEN switch with this rformat unless you redefine the Columns Per Line to 80 or less.

-n- Records have been scanned so far

Query displays this message after each 50 records scanned until it finds the first record to be SELECTed.

-n- records will be sorted

The activated rformat requires use of the SORT facility. This message is just letting you know that the sort is taking place and you may have a wait until the data is displayed.

RFORMAT Command

You'll get the following error messages when you issue an RFORMAT command. Rformat syntax errors are listed later.

Command does not take an argument

You cannot use an argument if you've used the /DISPLAY, /SPEED, or /LINEDIT switch with the RFORMAT command.

Error in PROC
< error message >

You used a /SPEED or /LINEDIT switch and Query was unable to create the text editor as a son process. See your system manager.

---x--- is not a defined format
There is no current qformat selected

You tried to activate an rformat that contains a QFORMAT statement which either selects an invalid qformat or uses a misspelled qformat name. Correct the error in the rformat definition.

---x--- is not a defined format
There is no current rformat selected

You've attempted to activate an rformat that is not in the .RFORMS file, has syntax errors, or you've misspelled the name.

There is no current rformat selected

You specified an rformat with errors. It's unacceptable. Fix the errors before activating the rformat again.

QFORMAT Command

You'll get the following error messages if you inaccurately use the QFORMAT command. Qformat syntax errors are listed later.

---x--- is not a defined format

You've attempted to activate a qformat that is not in the .QFORMS file, has syntax errors, or you've misspelled the name.

Command does not take an argument

You can not use an argument if you've used the /DISPLAY, /SPEED, or /LINEDIT switch with the QFORMAT command.

Error in PROC
< error message >

You used a /SPEED or /LINEDIT switch and Query was unable to create the text editor as a son process. See your system manager.

Miscellaneous Query Command Errors

You'll receive the following error messages if you inaccurately use one of the other general Query commands. The messages are arranged by command in alphabetical order.

CLI Command

Error in PROC
< error message >

Query was unable to create CLI as a son process. See your system manager.

HELP Command

No help available on topic:

The Query doesn't have any HELP information on the topic you requested. You may have made a typo or spelling error. Check it out. If you are not sure which topics do have HELP information, ask the Query by typing HELP or HELP OVERVIEW.

LEVELMARK Command

The key level seperator must be a single character

You tried to declare a key level separator with more than one character.

Qformat Syntax Errors

Query checks the syntax of all qformats when it first comes up or after a CLI, QFORMAT/SPEED, or QFORMAT/LINEDIT command. If there is a syntax error in a qformat, you'll receive any of the following messages before the first Query prompt appears on the terminal screen.

START_FORMAT must have a qformat name - on line --x--

There's an error in the START_FORMAT line. Have you specified a valid qformat name?

Looking for a START_FORMAT on line --x--

The Query can't find a valid START_FORMAT line. Check your syntax.

Bad START BYTE on line --x--

Query doesn't like the starting byte you specified on the named line. Be sure to use a valid number (a positive integer).

Bad END BYTE on line --x--

Query doesn't like the ending byte you specified on the named line. Be sure to use a valid number (a positive integer) greater than or equal to the start byte.

START BYTE is greater that END byte on line --x--

The first byte on the named line is larger than the last byte.

Bad KEY TYPE on line --x--

You specified an invalid key type on the named line. Check Chapter 4 of this manual for valid key types, or type HELP QFORMS_SYNTAX.

Too many Field Descriptor Lines in qformat.

You've used too many field descriptor lines in the qformat. The maximum allowed is 30.

--x-- is not a valid format

This message follows all other syntax error messages as a reminder that the named qformat is not valid.

Too many qformats in .QFORMS file

You've defined too many qformats for one .QFORMS file. One .QFORMS

Report Writer Runtime Errors

The following errors show up for certain individual records on a report. You'll get them after using a /RFORMAT switch with the READ or SELECT command in an attempt to use the rformat to display data.

The input record is not long enough

The Report Writer is trying to find data in fields which would be beyond the end of the record. Check your qformat and data.

CIS fault:

You defined a field in your qformat as a non-ASCII data type. The data the Report Writer found in that field is not consistent with the non-ASCII data type. Check your data and qformat.

Significant digits truncated:

Your number won't fit in the defined or default picture clause. The Report Writer truncated the extra digits. Check your picture clause.

Invalid numeric value:

You've used a Picture clause for a non-numeric field or else there is non-numeric data in what should be a numeric field. Check your data.

Divide by zero error. Zero result assigned:

Your DEFINE line included a division operation and on this input record the divisor for the DEFINE is a zero. Division by zero is undefined. The Report Writer uses this message to to let you know that this has happend and that it set the def-item to zero.

Rformat Syntax Errors

You'll get rformat syntax errors when you run RWCHECK or when you attempt to activate the rformat. The errors must be fixed before the Query will let you activate the rformat.

General Rformat Syntax Errors

You'll receive the following messages if syntax errors occur in any of the format descriptor lines that make up an rformat.

Not a valid command line

The Report Writer doesn't recognize a format descriptor line. Check the syntax. All commands must be delimited by a tab.

Wrong number of arguments for this command: -----

Check what you typed. You may have used blanks instead of tabs or the wrong number of arguments.

Too many arguments on one report command line

You used more arguments than allowed. Check the syntax.

Command line item is too large, line ignored

You specified an argument longer than 50 characters. Only comments may have items which are longer than 50 characters.

Invalid RFORMAT command: -----

The format descriptor line you entered isn't acceptable. Check the spelling and make sure that a tab delimits each field.

Ambiguous RFORMAT command definition: -----

The Report Writer doesn't recognize the command; the abbreviation is not unique.

There MUST be a detail, total, or break line somewhere.

You tried to define an rformat without at least one DETAIL line, one BREAK line, or one TOTAL line. The Report Writer won't accept it.

Field starting location is before beginning of line -----
A field spreads across the right end of -----
Overlapping items on -----

You'll get one of these error messages if the item(s) on the named line do not fit properly. Check the syntax. For instance, remember that an unpictured number requires a 14 character field.

Total Errors: -----

You've got the displayed number of errors in your rformat. Fix them.

BREAK Statement

Not a Data_item: -----

The Report Writer did not recognize the specified item. Check for typos, missing or misplaced tabs, and make sure that your qformat command worked. If you intended the item to be a literal, did you put quotes around it?

Too many unique BREAK items requested

You tried to define too many BREAK actions. You can define BREAK actions for up to 5 data items in the rformat.

Invalid PRINT_OP expression: -----

The Report Writer can't understand the expression. If you wanted to underline, parentheses must surround the named item. Note, also, that in a PRINT_OP expression, a blank seperates the keyword from the argument.

Item does not exist: -----

The operand you specified is not a data item, a defined item, or a literal. These are your only choices.

This field not yet used in a DETAIL line: -----

You wanted an item underlined, but you haven't named that item on a DETAIL line yet. Be sure the DETAIL statement for that item preceeds the BREAK statement for it.

Line location is invalid: -----

You specifed an invalid location for the BREAK information. You can use either a positive or negative number, or C, for centered.

Invalid BREAK or TOTAL expression: -----

The Report Writer doesn't understand what BREAK or TOTAL actions you want performed. Check the syntax and/or for typos.

COLUMNS PER LINE Statement

Invalid number: -----

You've used something other than a number as an argument; the COL/LIN statement requires a number as an argument.

Number too large: -----

You used a number as an argument, but it was too large. You can indicate up to 132 characters to a line in the COL/LIN statement.

Multiple CPL (characters per line) definitions

You tried to use more than one COL/LIN statement in the rformat. Only one is allowed.

DEFINE Statement

Too many DEFINE items specified

You tried to DEFINE too many items. You're limited to 10 DEFINE variables per rformat.

Item already exists: -----

You tried to re-define an item that has already been defined. Use a different variable name.

The expression is missing an operator

The Report Writer is looking for an operator separated from the two operands by spaces. Tabs will not work here.

Item does not exist: -----

The operand you specified is not a data item, a defined item, or a literal. These are your only choices.

DETAIL Statement

Invalid number: ----- Line # is too great: -----

Either of these messages means you specified an invalid DETAIL line number. Try an integer between 1 and 5.

Line location is invalid: -----

You specified an invalid location for the starting position of some DETAIL information. You can use a positive or negative number, or C, for centered.

Item does not exist: -----

The Report Writer doesn't recognize the named item. It is not a known item or a literal.

HEADER Statement

Invalid number: ----- Line # is too great: -----

Either of these messages means you specified an invalid HEADER line number. Try an integer between 1 and 10.

Line location is invalid

You specified an invalid location for the starting position of some HEADER information. You can use a positive or negative number, or C, for centered.

Not a Data_item: -----

You specified a data item that has not yet been defined in a qformat. Try a different data item, or define this one in a qformat.

LINES PER PAGE Statement

Invalid number: ----- Too few lines per page specified

Either of these messages means you specified an invalid number of lines. The minimum allowed is 30.

Multiple LPP (lines per page) definitions

You tried to use more than one LIN/PG statement in the rformat. Only one is allowed.

PICTURE Statement

Item does not exist: -----

You specified a picture clause for an item that doesn't exist. The named item must be a data or defined item. Try again.

Multiple PICTURE definitions for one item: -----

You can only specify one PICTURE per item. Choose the PICTURE which best fits. If necessary, DEFINE another item in terms of the first and then specify the PICTURE it needs.

Invalid characters in expression

You used nonpicture characters in your Picture clause. Check Chapter 5 in this manual for valid characters.

PICTURE definition is tooo long

You tried to use more than 30 characters.

Invalid PICTURE format: -----

You used invalid syntax in your Picture clause. Check the syntax.

Too many digits to the left of the decimal.
Too many digits to the right of the decimal.

Either of these messages means you used too large a Picture clause. You are allowed 12 characters to the left of the decimal point and 4 to the right.

QFORMAT Statement

Multiple qformat definitions

You specified more than one qformat in the rformat. Only one is allowed.

---x--- is not a defined format

You've specified a qformat that is not in the .QFORMS file, has syntax errors, or you've misspelled the name.

SORT Statement

Not a Data_item: -----

The named item must be a data item as defined in a qformat.

Too many SORT items specified

The current limit of items to be sorted has been exceeded.

Unknown SORT order requested: -----

The SORT statement takes either 'ASCENDING' or 'DESCENDING' or appropriate abbreviations for the order. Ascending order is the default.

TOTAL Statement

Invalid number: -----
Line # is too great: -----

Either of these messages means you specified an invalid TOTAL line number. Try an integer between 1 and 10.

Line location is invalid: -----

You specifed an invalid location for the TOTAL information. You can use either a positive or negative number, or C, for centered.

Item does not exist: -----

The Report Writer doesn't recognize the named item. Try either a data or defined item.

Invalid TOTAL expression: -----

The Report Writer assumes that you want total information displayed from the first three fields of the TOTAL statement, but can't interpret the fourth field. Check it out and try again.

End of Appendix

Index

Within this index, "f" or "ff" after a page number means "and the following page" or "pages". In addition, primary page references for each topic are listed first.

) (AOS CLI prompt) 1-5
> > (Query/Report Writer prompt) 1-5ff, 4-6

A

abbreviations 1-5
AOS 1-4f
    batch mode 3-15f
    Command Line Interpreter (CLI) 1-5
    SORT utility 5-5
    text file 3-1, 3-7, 4-1, 5-1
approximate key search 2-3, 2-5
argument 1-5, 2-15f
    ALL 1-5, 2-16
    for CONDITION command 2-15
    numeric 1-5, 2-6, 2-10, 2-16
ASCII characters 4-4, 3-6, 4-1

B

batch mode 3-15f, 5-8
    running macros in 3-14ff
BREAK
    actions 5-5f, 3-11
    information 5-6
    keyword 5-6
    line 5-5ff, 3-9ff, 5-2
BYE command 6-2, 1-8, 2-20, 6-1
bytes 3-2
    end 3-2, 2-8, 3-1, 4-1f
    start 3-2, 2-8, 4-1f

C

COBOL
    program 4-4
    picture clauses 4-5
CLI 1-5f, 3-6f, 3-13, 3-15f, 4-1, 5-1, 5-7f
    command 6-2, 1-8, 2-20, 6-1
    prompt 1-5
COL/LIN line 5-2f, 3-9f, 3-12
column headers 3-10
columns per line 5-3
Command Line Interpreter (See CLI)
commands
    arguments 1-5
    BYE 6-2, 1-8, 2-20, 6-1
    CLI 6-2, 1-8, 2-20, 6-1
    CONDITION 6-3, 1-8, 2-13, 2-15, 2-19, 3-15, 6-1
    DOWN 6-5, 1-7, 2-6, 6-1
    display 2-7, 1-8
    FORMAT 6-13, 2-7
    HELP 6-6, 1-8, 2-19, 6-1
    KEY 6-8, 1-7, 2-4f, 3-15, 6-1
    LEVELMARK 6-9, 1-8, 2-19, 6-1
    movement 2-3, 1-7
    NEXT 6-10, 1-7, 2-6, 6-1
    PATH 6-11, 1-7, 2-4f, 2-13, 6-1
    PRIOR 6-12, 1-7, 2-6, 6-1
    QFORMAT 6-13, 1-8, 2-7f, 2-13, 3-15, 6-1
    QUERY 6-1
    READ 6-14, 1-8, 2-7ff, 3-10ff, 4-6, 5-3ff, 5-11, 6-1
    RFORMAT 6-17, 1-8, 2-8f, 3-7, 3-15, 5-11, 6-1
    SELECT 6-19, 1-8, 2-7, 2-13ff, 3-1, 3-13ff, 4-6, 5-3ff, 6-1
    selection 2-7, 1-8
    switches 1-5
        /AND 2-15, 6-3
        /APPROX 2-5, 6-8
        /B 5-9, 3-15, 6-1
        /CLEAR 2-8, 2-8, 2-10, 2-15, 6-3, 6-13, 6-17
        /COUNT 2-16, 6-19
        /DISPLAY 2-8f, 6-13, 6-17
        /L 1-6, 2-12, 2-19, 6-6ff
        /L=filename 1-6, 2-12, 2-19, 6-6ff
        /LINEDIT 3-1, 3-6f, 3-16, 4-1, 4-6, 5-1, 6-13, 6-17
        /N 5-9
        /NAMES 2-8f, 3-6, 6-13, 6-17
        /NOFORMAT 2-16, 6-19
        /OR 2-15, 6-3
        /Q 3-13, 3-15, 6-1
        /R 5-2, 5-9
        /RETURN 2-11, 6-14, 6-19
        /RFORMAT 2-8ff, 3-10ff, 5-7, 5-11, 6-14, 6-19
        /S 5-9
        /SCREEN 2-13, 2-19, 3-10, 5-3, 6-14, 6-19
        /SPEED 3-1, 3-6f, 3-16, 4-1, 4-6, 5-1, 6-13, 6-17
        /SUBINDEXES 2-6f, 6-5, 6-10, 6-12
        /T 6-1
        /TRAVERSE 2-10, 2-16, 6-14, 6-19
        /UPTO 2-11, 6-14

syntax 1-6
UP 6-22, 1-7, 2-6, 6-1
COMMENT line 3-9f, 5-2
computations 3-10
condition 1-3, 1-8, 2-7, 2-13, 2-16, 3-17, 4-2, 1-3, 1-8
in macro 3-15
CONDITION
command 6-3, 1-8, 2-13, 2-15, 2-19, 3-15, 6-1
argument 2-15
switches
/AND 6-3
/CLEAR 6-3
/OR 6-3
statement 2-15, 3-1, 3-18, 3-4, 3-7, 4-4
control characters
CTRL-C CTRL-A 1-7
CTRL-U 1-7
correcting errors 1-7
current index 2-4

D

data
items 5-2, 3-11
type 4-1
database 3-2, 1-1
records 1-3, 2-13, 1-8, 2-10
datanames 2-15
datatype string 3-6
data_items 5-6f
DATE field 5-3
DEFINE
keyword 5-4
line 5-4, 3-11f, 3-9, 5-2
defined data item 3-12
def_items 3-12, 5-2
delete key 1-7
DESCENDING keyword 5-5
description 2-8
designing
macros 3-1ff
qformats 3-1ff
rformats 3-1ff
DETAIL
information 5-3f, 3-9ff
keyword 5-4
line 5-4ff, 3-9ff, 5-2
display
commands 2-7, 1-8
mode 2-8
records 2-7, 1-7, 3-18
document 3-10
documentation conventions iv
DOWN command 6-5, 1-7, 2-6, 6-1
switch
/SUBINDEXES 6-5

E

end byte 3-2, 2-8
integer 4-2
END_FORMAT
keyword 3-6
line 4-1, 3-1, 3-9
END_REPORT 5-2
keyword 5-7
line 5-7, 3-9ff, 5-1
entry 1-1, 2-3
errors
correcting 1-7
messages B-1
runtime format 4-6

F

field 1-1f, 1-4, 2-7f, 3-2, 3-6, 4-2, 5-2
description types 4-4
descriptions
end byte 4-1f
start byte 4-1f
title 4-1f
type 4-1f
name 3-1, 4-1
field descriptor lines 4-1f, 3-1
files
extensions, .QU 3-14
.QFORMS 3-1, 2-8, 3-6f, 3-10, 4-1f, 4-6, 5-1
syntax 4-1
.RFORMS 5-1, 3-7, 3-10ff, 5-9ff
FORMAT command 6-13, 2-7
switches
/CLEAR 6-13
/DISPLAY 6-13
/L 6-13
/L=filename 6-13
/LINEDIT 6-13
/NAMES 6-13
/SPEED 6-13
format descriptor line 3-9ff, 5-1f, 5-7
full record format 3-2ff

G

generic key search 2-3f

H

HEADER
information 5-3, 3-9ff
keyword 5-3
line 5-3, 3-9, 3-12, 4-3f
HELP command 6-6, 1-8, 2-19, 6-1
switches
/L 6-6
/L=filename 6-6
horizontal movement 2-6

I

index 3-18, 1-3, 2-13
  current 2-4
  level 2-4
  structure 1-1, 1-7, 2-3ff, 2-10, 2-16
information, summary 5-5ff, 3-10f
INFOS
  file 2-3, 1-1
    records 4-1
  index 4-1
  system database 2-1
items, summary 5-6

K

key 1-1, 2-3ff, 2-10, 3-11, 3-18
  fragments 2-4f
  search 2-3f
KEY command 6-8, 1-7, 2-4f, 3-15, 6-1
  switches
    /APPROX 6-8
keyboard 1-6
keywords 5-6
  DEFINE 5-4
  DESCENDING 5-5
  DETAIL 5-4
  END_FORMAT 5-2
  END_REPORT 5-7
  HEADER 5-3
  LIN/PAGE 5-2
  PAGE_EJECT 5-6
  PICTURE 5-4
  POST_BREAK_SPACE 5-6
  PRE_BREAK_SPACE 5-6
  QFORMAT 5-2
  SORT 5-5
  START_FORMAT 3-6, 4-1
  START_REPORT 5-2
  TOTAL 5-6
  UNDERLINE 5-6

L

LEVELMARK 2-19, 1-8
  command 6-9, 1-8, 2-19, 6-1
lines per page 5-2
LIN/PAGE
  keyword 5-2
  line 5-2f, 3-9ff
logging on 1-4
lowercase 1-6

M

macros 3-13ff, 1-4
  path in 3-15
  running in batch mode 3-14ff
movement commands 2-3, 1-7

N

NEXT command 6-10, 1-7, 2-6, 6-1
  switch
    /SUBINDEXES 6-10
non-ASCII formatting 4-2ff
numeric
  arguments 2-6, 2-10, 2-16
  data 5-4
  item 3-11

O

optional data type 4-2

P

PAGE field 5-3
PAGE_EJECT keyword 5-6
path 1-1, 2-3ff, 2-10, 2-16, 3-15ff
  in macro 3-15
PATH command 6-11, 1-7, 2-4f, 2-13, 6-1
PICTURE 5-2
  characters 5-5
  keyword 5-4
  line 5-4, 3-9ff
pointers 1-1
POST_BREAK_SPACE keyword 5-6
PRE_BREAK_SPACE keyword 5-6
PRIOR command 6-12, 1-7, 2-6, 6-1
  switch
    /SUBINDEXES 6-12
prompt
  AOS CLI 1-5
  Query/Report Writer 1-5ff, 4-6

Q

qformat 2-7f, 1-4, 2-10, 2-15f, 2-21, 3-1ff, 4-1ff, 5-1f
  error messages 4-6
  file 4-1ff
  in macro 3-15
  name 4-1
QFORMAT
  command 6-13, 1-8, 2-7f, 2-13, 3-15, 6-1
    switches
      /CLEAR 6-13
      /DISPLAY 6-13
      /L 6-13
      /L=filename 6-13
      /LINEDIT 6-13
      /NAMES 6-13
      /SPEED 6-13
  keyword 5-2
  line 5-2, 3-9ff
QUERY command 6-1
  switches
    /B=filename 6-1
    /Q=filename 6-1
    /T=filename 6-1

Query
commands 6-1ff, 1-7, 2-2
facility 1-1
macro 1-4
prompt 1-5ff
sessions 3-14
tasks 2-2
quotation marks 1-6, 2-15, 3-11

R

READ command 6-14, 1-8, 2-7ff, 3-10ff, 4-6, 5-3, 5-7, 5-11, 6-1
switches
/L 6-14
/L=filename 6-14
/RETURN 6-14
/RFORMAT 6-14
/SCREEN 6-14
/TRAVERSE 6-14
/UPTO 6-14
records 5-5, 1-1f, 1-4, 1-7, 2-7, 2-15f, 3-1f
format 1-2
structures 3-2
relational operators 2-15
Report Writer 5-1ff, 1-1f, 2-8, 2-13, 2-19, 3-9f
reports 5-1ff, 1-3, 1-8, 2-8, 2-13, 2-19, 3-7, 3-11ff
format 5-1ff, 1-1f, 2-8
forms 5-1
sample page of 5-8ff
shape of 3-10
size of 3-10
rformat 5-1f, 1-1ff, 1-8, 2-8ff, 2-18f, 2-21, 3-7ff, 3-17, 5-7
definition 5-8
in macro 3-15
syntax 5-1, 1-3, 3-12, 5-7
RFORMAT command 6-17, 1-8, 2-8f, 3-7, 3-15, 5-11, 6-1
switches
/CLEAR 6-17
/DISPLAY 6-17
/L 6-17
/L=filename 6-17
/LINEDIT 6-17
/NAMES 6-17
/SPEED 6-17
route 1-1, 1-7
runtime format errors 4-6, B-1ff
RWCHECK 5-8ff, 1-3, 2-21, 3-7, 3-12, 5-2
switches
/A 5-9
/L 5-9
/N 5-9
/R 5-9
/S 5-9

S

sample page of report 5-8ff, 1-3, 3-7, 3-12
SELECT command 6-19, 1-8, 2-7, 2-13ff, 3-13ff, 4-6, 5-3, 5-7ff, 6-1
switches
/COUNT 6-19
/L 6-19
/L=filename 6-19
/NOFORMAT 6-19
/RETURN 6-19
/RFORMAT 6-19
/SCREEN 6-19
/TRAVERSE 6-19
selecting records 1-7, 3-18
selection
commands 2-7, 1-8
criteria 2-7, 1-1
shape of report 3-10
size of report 3-10
SORT
AOS utility 5-5
key 5-5
keyword 5-5
line 5-5f, 3-9ff
Stand-Alone Compiler 5-8ff, 1-3, 2-21, 3-12
start byte 3-2, 2-8, 3-6
integer 4-2
START_FORMAT
keyword 4-1, 3-6
line 4-1f, 3-1, 3-9
START_REPORT
keyword 5-2
line 5-1f, 3-9ff, 5-7
string literal 1-6
subindex 1-1, 2-3ff, 2-13, 2-16, 3-18, 5-5
level 2-6
summary
information 5-5ff, 3-10f
items 5-6
switches
/AND 2-15, 6-3
/APPROX 2-5, 6-8
/B 5-9, 3-15, 6-1
/CLEAR 2-8, 2-10, 2-15, 6-3, 6-13, 6-17
/COUNT 2-16, 6-19
/DISPLAY 2-8f, 6-13, 6-17
/L 1-6, 2-12, 2-19, 6-6ff
/L=filename 1-6, 2-12, 2-19, 6-6ff
/LINEDIT 3-1, 3-6f, 3-16, 4-1, 4-6, 5-1, 6-13, 6-17
/N 5-9
/NAMES 2-8f, 3-6, 6-13, 6-17
/NOFORMAT 2-16, 6-19
/OR 2-15, 6-3
/Q 3-13, 3-15, 6-1
/R 5-2, 5-9
/RETURN 2-11, 6-14, 6-19

/RFORMAT 2-8ff, 3-10ff, 5-7, 5-11, 6-14, 6-19
/S 5-9
/SCREEN 2-13, 2-19, 3-10, 5-3, 6-14, 6-19
/SPEED 3-1, 3-6f, 3-16, 4-1, 4-6, 5-1, 6-13, 6-17
/SUBINDEXES 2-6f, 6-5, 6-10, 6-12
/T 6-1
/TRAVERSE 2-10, 2-16, 6-14, 6-19
/UPTO 2-11, 6-14
syntax 3-7
errors 5-2
rules 3-12

T

temporary variables 5-14, 3-11
text editor 3-1, 3-7, 3-14f, 4-1, 5-1
text file 3-13, 4-1, 5-1
TOTAL
information 5-6f, 3-10
keyword 5-6
line 5-6f, 3-9ff

U

UNDERLINE keyword 5-6
UP command 6-22, 1-7, 2-6, 6-1
uppercase 1-6

V

vertical movement 2-6

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title _____ No. _____

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- ☐ EDP Manager
- ☐ Senior System Analyst
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ Other _____

What programming language(s) do you use? _____

_____

How Do You Use This Manual?

(List in order: 1 = Primary use)

- _____ Introduction to the product
- _____ Reference
- _____ Tutorial Text
- _____ Operating Guide
- _____ _____

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name _____ Title _____ Company _____

Address _____ Date _____

SD-00742

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
C/350, C/330, C/300		
S/250, S/230, S/200		
S/130		
AP/130		
CS Series		
N3/D		
Other NOVA		
microNOVA		
Other (Specify) _____		

3. Software

- ☐ AOS
- ☐ DOS
- ☐ SOS
- ☐ RDOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ Algol
- ☐ DG/L
- ☐ Cobol
- ☐ ECLIPSE Cobol
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ Interactive
- ☐ Fortran
- ☐ RPG II
- ☐ PL/1
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ RSTCP
- ☐ HASP
- ☐ RJE80
- ☐ SAM
- ☐ CAM
- ☐ 4025
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

DG-05810

DataGeneral

Data General Corporation, Westboro, Massachusetts 01581, (617) 366-8911

FOLD FOLD

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator

Southboro, Massachusetts 01772