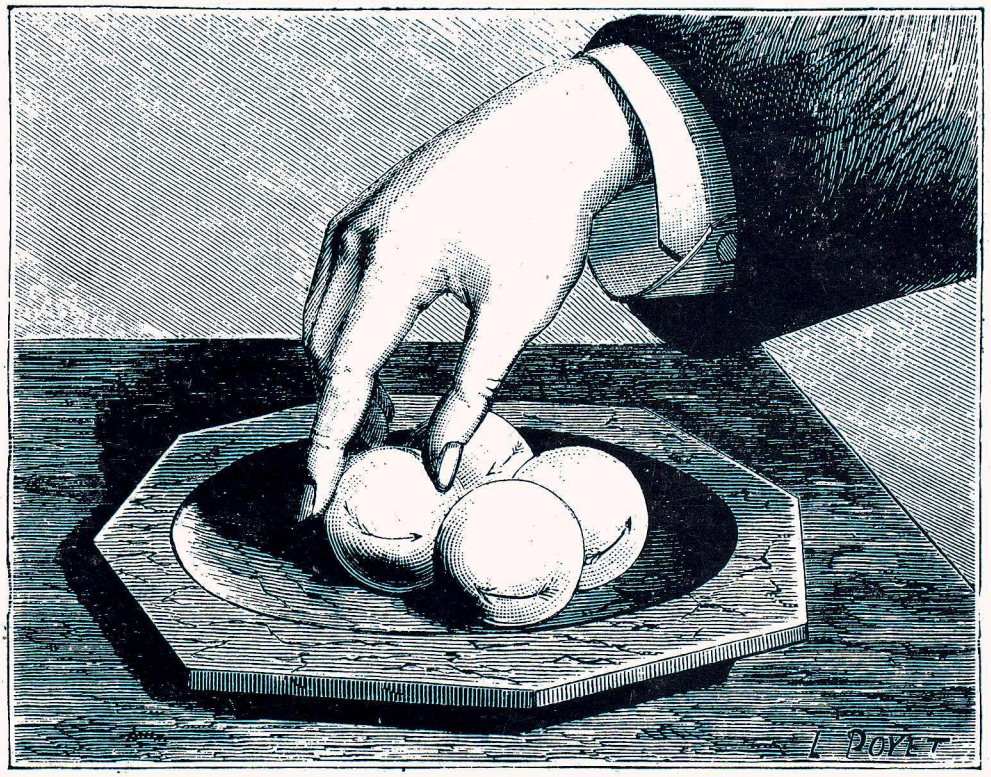
The Pick Series



Pick ACCESS

A GUIDE TO THE SMA/RETRIEVAL LANGUAGE

O'Reilly & Associates, Inc.

The Pick Series

Pick ACCESS

A GUIDE TO THE SMA/RETRIEVAL LANGUAGE

By Walter Gallant
and the Staff of O'Reilly & Associates, Inc.

O'Reilly & Associates, Inc.

First Edition, Nov. 1989

ISBN 0-937175-41-2

The Pick Series

COMPLETE, ACCESSIBLE GUIDES TO PICK

The Pick Series offers user-oriented documentation—helping new users learn about Pick quickly and helping experienced users find accurate information easily. The Pick Series is a complete documentation set for all users. It tackles the Pick system at a level of depth not found elsewhere. Books in the series are based on a mature implementation of the Pick operating system (R83) with notes on SMA standards and specific differences among major Pick implementations.

TECHNICAL EDITOR

W. Clifton Oliver, CCP

SERIES EDITOR

Dale Dougherty

The Pick Series

The goal of the Pick Series is to provide Pick documentation that is user-oriented: to help new users learn about Pick quickly and to help experienced users find accurate information easily. These books are written in a conversational tone, with lots of examples, as if an experienced user were by the reader's side. The Pick Series offers complete documentation of the Pick operating system (R83) with notes on SMA standards and specific differences among major Pick implementations.

Books in the series include:

- *Pick ACCESS: A Guide to the SMA/RETRIEVAL Language*
  (Available 12/89, ISBN 0-937175-41-2, $29.95)
- *A Guide to the Pick System*
  (Available 1/90, ISBN 0-937175-43-9, $34.95)
- *Pick BASIC: A Reference Guide*
  (Available 2/90, ISBN 0-937175-42-0, $39.95)
- *Master Dictionary Reference Guide: User Account Verbs*
  (Available 2/90, ISBN 0-937175-44-7, $39.95)
- *System Administration: A Guide to Managing the Pick/SMA Operating System*
  (Available 3/90, ISBN 0-937175-45-5, $34.95)
- *SYSPROG Reference Guide: SYSPROG Account Verbs*
  (Available 3/90, ISBN 0-937175-46-3, $29.95)
- *PROC: A Guide to the PROC Processor*
  (Available 4/90, ISBN 0-937175-47-1, $21.95)

O'Reilly & Associates, Inc.

632 Petaluma Avenue • Sebastopol, CA 95472 • 800-338-6887
in CA 800-533-6887 • local /overseas +1- 707-829-0515

C O N T E N T S

List of Chapters

CONTENTS

PREFACE

About This Book

Pick ACCESS is a complete introduction and guide to the Pick/SMA retrieval language. This retrieval language, used on all Pick and Pick-related systems, is known by a variety of names: ACCESS (Pick Systems), INFO/ACCESS (ADDS Mentor), RECALL (Ultimate), ENGLISH (Reality), RETRIEVE (uniVerse), INFORM (Prime INFORMATION), etc. Although there are some differences in the ways different manufacturers have implemented ACCESS on their systems, for the most part the ACCESS processor works in a similar manner on all Pick systems.

ACCESS is perhaps the most familiar and easy-to-use feature of the Pick system. It is an integral part of the system, with the power and flexibility to create custom reports that would otherwise require special programming to create. ACCESS is easy to use because it allows users to enter natural-language "sentences" (such as "SORT DATA IN THE CUSTOMERS FILE BY LAST-NAME") to query the database. It can be used for both regular periodic reporting as well as for *ad hoc* reports. No programming knowledge is necessary in order to use ACCESS.

In this book, whenever we refer to "the Pick operating system" or to "Pick systems," we do not refer exclusively to the operating system developed and marketed by Pick Systems, Inc., but rather to Pick systems generally. Although different manufacturers market versions of the Pick system under their own trade names, such as Mentor (ADDS), Ultimate, uniVerse (VMark), etc., they are all versions of what has become known generically as the Pick system.

The Pick system has clearly withstood the test of time. The same cannot be said for the standard Pick documentation, which many users find difficult to use, outdated, and inaccessible. This book provides Pick documentation that is user-oriented: with *Pick ACCESS*, new users will be able to learn about ACCESS quickly, and experienced users will find accurate information easily.

This book aims at completeness. While the information in *Pick ACCESS* is fully compatible with the *SMA/RETRIEVAL Language Specification* published in January 1988 by SMA, we have tried, in addition, to be as inclusive as possible. This book is not just about those elements of ACCESS that are common to all systems; it also takes note of significant differences found among the different implementations. For example, Chapter 7 discusses forms generation (which is not included in the SMA standards) and points out the different ways systems such as ADDS Mentor and Ultimate implement this feature. We also highlight the ways systems such as Prime INFORMATION and VMark's uniVerse differ from main-line Pick systems. The reader should not expect, however, to find every detail of every Pick system completely documented here. For full details about your own system, you will still need to consult the reference manuals provided by your system supplier.

Pick ACCESS is the first book in O'Reilly & Associates's user-oriented series of complete, accessible guides to the Pick system. The Pick Series offers a complete Pick documentation set for both new and experienced users. It is based on a mature implementation of the Pick R83 operating system, follows the SMA standards, and notes specific differences among major Pick implementations. Forthcoming books in the series will include:

- *A Guide to the Pick System.*
- *Pick BASIC: a Reference Guide.*
- *System Administration: A Guide to Managing the Pick/SMA Operating System.*
- *Master Dictionary Reference Guide: User Account Verbs.*
- *SYSPROG Reference Guide: SYSPROG Account Verbs.*
- *PROC: A Guide to the PROC Processor.*

Summary of Contents

Pick ACCESS has two parts. The first part, in eight chapters, provides a complete introduction to ACCESS. The first four chapters, written in a tutorial style and including lots of examples, give the new user a general overview of query syntax . Subsequent chapters explore some of the more specialized aspects of ACCESS such as forms generation and the use of processing codes (correlatives and conversions).

The second part, in four appendixes, contains alphabetic reference guides to the verbs, keywords, and processing codes used by the ACCESS processor. Complete syntax for each verb, keyword, and code is given, along with explanations of all syntactical parameters and options. Appendix D provides an outline of Pick dictionary structures, summarizing the two main types of Pick and Pick-related file dictionaries, Pick/SMA, and uniVerse/Prime INFORMATION.

Assumptions

We assume that a Pick system has already been installed on your computer and that you have some familiarity with the Terminal Control Language (TCL). We also assume you have a basic understanding of Pick file structure: that is, that you know about file dictionaries and how they define the structure of the data in a file. Even if you don't know much about file dictionaries, however, *Pick ACCESS* will not be difficult to use to retrieve data and generate reports from files that have already been set up on your system. You will find a complete explanation of the structure and functions of Pick files in *A Guide to the Pick System*, another book in the Pick series (see also Appendix D in this book, "File Dictionary Structures").

How to Use This Manual

Pick ACCESS is divided into eight chapters and four appendixes.

Chapter 1, "An Overview of ACCESS," introduces the ACCESS processor and the fundamental elements of its query language.

Chapter 2, "ACCESS Syntax," is a general description of the syntax of ACCESS. Each parameter that can be included in an ACCESS query is presented, accompanied by examples of its use. Also included is an explanation of how ACCESS processes different types of query, along with a description of default processing. The chapter ends with brief descriptions of all ACCESS verbs and keywords.

Chapter 3, "Producing Reports with LIST and SORT," shows how to produce many different types of report using the LIST and SORT verbs. Selection expressions, sort expressions, and print limiters are explained in detail, with many examples.

Chapter 4, "Formatting Reports," shows how to format ACCESS reports using keywords of different types. Topics include customizing headings and footings, calculating totals and subtotals, producing control breaks, etc.

Chapter 5, "Using Select-Lists," describes and explains how to generate, use, save, and retrieve select-lists. Select-lists allow you to identify and select only those items in a file that you want to process.

Chapter 6, "Specialized Processing," describes and explains how to use ACCESS verbs that let you process data in specialized ways: creating labels in block format, generating statistics about data files, copying data to and from tape, and restructuring data.

Chapter 7, "Forms Generation," describes and explains how to generate reports to be printed on forms (invoices, checks, order forms, etc.).

Chapter 8, "Correlatives and Conversions," describes and explains in detail how to use correlative and conversion codes to enhance the processing and manipulation of data both within a file and among different files. Many examples of correlatives and conversions are included.

Appendix A, "ACCESS Commands," is a reference guide to all ACCESS verbs arranged in alphabetical order. Each entry includes a brief explanation of what the command does, a complete explanation of its syntax, and a description of how to use the command.

Appendix B, "ACCESS Keywords," is an alphabetic reference guide to all ACCESS keywords, with examples illustrating how to use them. Also

included are tables listing the keywords according to their different categories: connectives, relational operators, modifiers, and options.

Appendix C, "Correlative and Conversion Codes," is a reference guide to all of the ACCESS correlative and conversion codes arranged in alphabetical order. Each entry includes a brief explanation of what the code does, a complete explanation of its syntax, and a description of how to use the code.

Appendix D, "File Dictionary Structures," provides a summary of the dictionary structures of Pick data files. Included are descriptions of the standard Pick/SMA File and Attribute Definition items, as well as a description of the type of dictionary items found on Prime INFORMATION and uniVerse systems.

Conventions

We use the following conventions for indicating command line syntax.

Convention	Usage
BOLD CAPS	Anything shown in large bold characters must be typed exactly as shown.
italics	Anything shown in italics is variable information for which the user provides a specific value.
( )	Parentheses must be typed. It is usually sufficient to type only the first parenthesis; the second is optional.
[ ]	Anything shown enclosed in square brackets is optional. The square brackets themselves are not typed.
|	A vertical bar separating two or more elements indicates that any *one* of the elements can be typed.
{ }	If two or more elements are enclosed within curly braces and separated by a bar, one of the elements *must* be typed.

All punctuation marks included in syntax format lines (e.g., commas, parentheses, angle brackets, underscores, hyphens) are required in the syntax unless otherwise indicated. Square brackets are not typed.

In the following syntax example:

LIST [**DICT**] *filename* [**WITH** [**EVERY** | **EACH**] *attribute-name value-list*] [(**P**)]

the only two elements of the line that must be entered are "LIST" and "*filename*". "LIST" must be entered exactly as shown. "*filename*" is a variable; the user can enter the name of any accessible file. "*attribute-name*" and "*value-list*" are also variables that the user supplies. The vertical bar indicates that either "EVERY" or "EACH" can be entered; the brackets indicate that both of these keyboards are optional. If the "P" option is entered, it must be enclosed within parentheses.

When variables supplied by the user are two or more words long, hyphens are used instead of blank spaces to separate the words in order to show that only one element is required. For example, in the command:

LIST *filename item-list*

the word *filename* indicates a single element, and the words *item* and *list* joined by a hyphen likewise indicate a single element.

Notation in Examples

In screen examples we use the following conventions:

Convention	Usage
BOLD	Anything the user types as input is shown in bold characters.
PLAIN	Any output displayed by the system (prompts, responses to user input, etc.) is shown in plain characters.
<RETURN>	Indicates that the RETURN key must be pressed.
<CTRL-*char*>	Indicates that a *control character* is to be typed. To enter a control character, simultaneously hold down the CONTROL (CTRL) key and press the specified character.

Most commands are entered by typing the command line at one of the prompts and then pressing the RETURN key. This is true whether you are entering a one-letter Editor command or a lengthy ACCESS statement.

Therefore, in examples the RETURN key symbol (**<RETURN>**) is shown only in cases where it is needed for clarity.

This symbol indicates an important note or caution.

Acknowledgements

We would like to thank everyone at Applied Digital Data Systems, Inc., for reviewing this material in its earlier life, and for the use of the ADDS Mentor 6000 system on which all of the examples in this book were generated. Special thanks to Robin White, Dave Yulke, Mike Hannigan, Joe Ferraro, Cliff Olsen, Linda Lutz, and many technical support people, all of whom contributed to the book in both large and small ways. Thanks, too, to Charles Cornell of VMark Software for his help over the years.

Thanks are also due to the O'Reilly & Associates staff who worked on the book: Julie Buckler and Dale Dougherty who helped to write it, and Michael Sierra, Laurel Erickson, and Will Hirshowitz who wrestled with Microsoft Word to produce it. Edie Freedman designed the covers.

We have made every effort to verify all the information in this book. Any errors that remain are our own.

How to Contact Us

To help us provide you with the best possible documentation, please write and tell us about any mistakes you find in this book or about how you think it might be improved.

Our U.S. mailing addresses are:

Ordering

O'Reilly & Associates, Inc.
632 Petaluma Avenue
Sebastopol, CA 95472
(800) 338-6887

Editorial (Walter Gallant)

O'Reilly & Associates, Inc.
90 Sherman Street
Cambridge, MA 02140
(617) 354-5800

CHAPTER 1

An Overview of ACCESS

ACCESS is a dictionary-driven database processor. It uses a query language that generates formatted reports from a database. ACCESS queries are made up of English-like words whose meanings match the functions they perform. Queries are entered either directly at the TCL prompt or through a proc or PICK/BASIC application program.

Chapter 1 provides a general introduction to ACCESS. Some examples of simple ACCESS queries are shown along with the reports they produce on the screen. You'll see how ACCESS does the following:

- Lists items in a file.
- Uses output specifications to determine what data to print or display in a report.
- Uses selection expressions to determine what data items are to be included.
- Uses sort expressions to determine the order in which data is printed or displayed.
- Uses keywords to format a report in different ways.

Chapter 2 explores in more detail the formal syntax of ACCESS queries.

Using ACCESS

ACCESS provides great flexibility in extracting information from a database and in formatting reports. This makes ACCESS a powerful tool for applications development.

ACCESS queries perform the following operations:

- Report data by displaying the contents of specified attributes.
- Extract data by defining the criteria by which items are selected.
- Format reports by controlling how the data is to be displayed.

The main component of an ACCESS query is an action-oriented *verb*, such as LIST, SORT, or COUNT. ACCESS verbs can be used to:

- Display or print file items.
- Sort items according to multiple keys.
- Generate forms.
- Count and produce statistics for selected items.
- Select items for further processing.
- Read and write data to tape.
- Display file hashing statistics.

LIST is a general-purpose verb that lists items in a file. By default, this list is displayed on the terminal screen. Let's look at an example using a file named CUSTOMERS. If you type the following statement and press the RETURN key, a listing such as the one shown in Example 1-1 is produced from the CUSTOMERS file.

This single-column report shows the item ID for each item in the CUSTOMERS file. (The item ID is a unique identifier that locates each item in a file.) Each report can have a heading, which typically contains the page number, time, and date, along with a footer, which typically displays the number of items included in the report.

```
>LIST CUSTOMERS

PAGE   1                                          10:43:27  01 NOV 1989

CUSTOMERS...

HJENK7129
JBOHA5422
JBROW6749
JBUCK6488
BLEAR6803
JMASO6378
AORLA5993
SPIRS5289
MASHX5777
AEDWA5224
JPEER5993
RPIER5539
AJOHN5396
HJOHN7265
HHIGG6849
DEDGE6635
BLAMP6196
AMEAD5619

18 ITEMS LISTED.
```

Example 1-1.
By default the LIST verb lists item IDs only.

ACCESS queries are usually not this simple. They generally include an *output specification*, which is a list of the attributes whose data is to be included in the report. For example, you could generate a report that displays the last name, street address, city, and state of all customers in the CUSTOMERS file. These are attributes (i.e., fields) containing data for items in the CUSTOMERS file.

The CUSTOMERS dictionary contains Attribute Definition items for LAST-NAME, STREET, CITY, and STATE. We can specify the names of these attributes in the query:

```
>LIST CUSTOMERS LAST-NAME STREET CITY STATE
```

The report that results is shown in example 1-2.

This report displays all of the items in the CUSTOMERS file, producing a line of data for each customer. The format of this report is columnar: the

contents of each attribute are listed vertically, and each customer item is listed horizontally. As described in Chapter 2, it is also possible to generate reports in noncolumnar format.

```
PAGE   1                                          10:43:46  01 NOV 1989

CUSTOMERS... Last Name. Street..... City..... State

HJENK7129    JENKINS    1222 MAIN STREET     INDIANAPOLIS    IN
JBOHA5422    BOHANNON   126 TREMONT STREET   BOSTON          MA
JBROW6749    BROWN      129 BOYLSTON STREET  BOSTON          MA
JBUCK6488    BUCKLER    26 STONE AVENUE      LINCOLN         IN
BLEAR6803    LEARY      34 TREMONT STREET    BOSTON          MA
JMASO6378    MASON      226 ROCK ROAD        LINCOLN         IN
AORLA5993    ORLANDO    55 VENTURA HIGHWAY   VENICE          CA
SPIRS5289    PIRS       112 APPLEBEE ROAD    WINSTON         NC
MASHX5777    ASH        912A E. OAK STREET   INDIANAPOLIS    IN
AEDWA5224    EDWARDS    51 BLAIR AVENUE      SUDBURY         MA
JPEER5993    PEERCE     89 RIALTO WAY        LOS ALTOS       CA
RPIER5539    PIERCE     123 W RIDGEWOOD AVE  RIDGEWOOD       NJ
AJOHN5396    JOHNSON    760 JEFFERSON STREET LOUISVILLE      KY
HJOHN7265    JOHNSON    45 50TH STREET       OMAHA           NB
HHIGG6849    HIGGINS    54 25TH STREET       OMAHA           NB
DEDGE6635    EDGECOMB   338 BROADWAY         MIAMI           FL
BLAMP6196    LAMPSON    344 TREMAIN ROAD     BOSTON          MA
AMEAD5619    MEADE      251 BLOWNEY AVENUE   SUDBURY         MA

18 ITEMS LISTED.
```

Example 1-2.

Output specifications determine what data is to be listed in a report.

By default, all ACCESS reports automatically include item IDs. You can, however, use the ID-SUPP modifier to suppress the inclusion of these item IDs in a report. There are many other ACCESS modifiers that affect the processing or format of a report. These modifiers are shown in a table at the end of Chapter 2. Formatting modifiers are described in Chapter 4.

The rest of this chapter demonstrates the power and flexibility of ACCESS by expanding the preceding report. All of the syntax elements treated here are described in detail in Chapter 2.

Selecting Data

To generate a report about a subset of the CUSTOMERS (or any other) file, use a *selection expression* to define criteria against which all items in the file will be compared. The report then includes only those items that meet the selection criteria.

For example, to select only those customers who live in the state of Indiana, you could add the following selection expression to the preceding ACCESS query:

```
WITH STATE = "IN"
```

This expression compares the data in the attribute STATE to the literal "IN". "IN" is enclosed in double quotes to indicate that it is a literal value.

The ACCESS query now looks like the one shown in Example 1-3.

```
>LIST CUSTOMERS WITH STATE = "IN" LAST-NAME STREET
  CITY STATE ID-SUPP

PAGE   1                                        10:44:25  01 NOV 1989

Last Name. Street.............. City........ State

JENKINS    1222 MAIN STREET     INDIANAPOLIS    IN
BUCKLER    26 STONE AVENUE      LINCOLN         IN
MASON      226 ROCK ROAD        LINCOLN         IN
ASH        912A E. OAK STREET   INDIANAPOLIS    IN

4 ITEMS LISTED.

>
```

Example 1-3.
A selection expression specifies what criteria must be met in order for items to be included in a report.

All the items listed in the report contain "IN" as the value of the attribute STATE.

The following examples illustrate how ACCESS queries closely follow the patterns of natural-language sentences:

```
LIST CUSTOMERS WITH LAST-NAME < "MARTIN"
LIST EMPLOYEES WITH SALARY > "25,000"
LIST CUSTOMERS WITH STREET = "56 SPRINGFIELD STREET"
LIST STUDENTS WITH FIRST-NAME LIKE "MARYLYNN"
```

Sorting Data

The report shown in Example 1-3 lists items in a somewhat random order, based on where they are located in the file. Let's assume, however, that you want to display customer names in alphabetical order. To specify the order of items in a report, you can use a *sort expression*. A sort expression specifies which attribute to sort by and whether to sort it in ascending or descending order. A SORT query consists of a verb with sorting capabilities, such as the SORT verb, one of the four BY modifiers, and the name of the attribute whose data will be sorted.

Here are some sample sort queries:

```
SORT CUSTOMERS BY LAST-NAME
SORT ORDERS BY-DSND AMOUNT
SORT ORDERS WITH AMOUNT > "250" BY AMOUNT
SORT CUSTOMERS BY STATE BY LAST-NAME
```

The following query sorts customers by their last names.

```
>SORT CUSTOMERS WITH STATE = "IN" BY LAST-NAME
  LAST-NAME STREET CITY STATE ID-SUPP
```

The resulting report is shown in Example 1-4.

Formatting a Report

As you've seen in the examples so far, an ACCESS report includes the following default heading:

```
PAGE n                                    time        date
```

This heading can be seen at the top of Example 1-4.

```
PAGE   1                                          10:44:56  01 NOV 1989

Last Name. Street..... City..... State

ASH         912A E. OAK STREET   INDIANAPOLIS     IN
BUCKLER     26 STONE AVENUE      LINCOLN          IN
JENKINS     1222 MAIN STREET     INDIANAPOLIS     IN
MASON       226 ROCK ROAD        LINCOLN          IN

4 ITEMS LISTED.

>
```

Example 1-4.
A sort expression determines the order in which items in a report are to be listed.

Reports generally also include an end-of-list message in the following format:

n ITEMS LISTED.

n is the number of items listed in the report. ACCESS queries can, however, contain specifications for more complex headings, footings, and other formatting parameters. The query shown in Example 1-5 includes a heading on each page of the report that reads "INDIANA CUSTOMERS".

```
>SORT CUSTOMERS WITH STATE = "IN" BY LAST-NAME
   LAST-NAME STREET CITY STATE ID-SUPP HEADING
   "INDIANA CUSTOMERS"
```

```
INDIANA CUSTOMERS
Last Name. Street..... City..... State

ASH         912A E. OAK STREET   INDIANAPOLIS     IN
BUCKLER     26 STONE AVENUE      LINCOLN          IN
JENKINS     1222 MAIN STREET     INDIANAPOLIS     IN
MASON       226 ROCK ROAD        LINCOLN          IN

>
```

Example 1-5.
The HEADING modifier lets you customize the report heading.

The default heading is replaced by the new heading. Also, notice that the usual end-of-list message does not appear when you define a heading.

Sending a Report to the Printer

All the reports we've seen so far have been displayed on the terminal screen. ACCESS reports can be sent to the printer by including the LPTR modifier or the P option at the end of a query. P is a parenthetical option, one of several which can be applied to ACCESS commands. For example:

```
>SORT CUSTOMERS WITH STATE = "IN" BY LAST-NAME
  LAST-NAME STREET CITY STATE ID-SUPP HEADING
  "INDIANA CUSTOMERS" (P)
```

This report is sent to the printer that has been assigned to the user's account. Because you may be sharing a printer with other users, the report may not be immediately output on the printer. Instead, it is "spooled" in a print queue, and the systems will send it to the printer in turn.

Reviewing the Syntax of a Query

Now that we have constructed a more complex ACCESS query, it might be useful to review the elements that make it up. Example 1-6 shows the query discussed in the preceding section broken down into its component parts.

The verb is the most important part of an ACCESS query. The verb specifies the basic operation required to generate a report. The syntax elements that follow the verb modify this basic operation.

An Overview of ACCESS Verbs

ACCESS verbs are words that specify the operation to be performed on a database. LIST and SORT are general-purpose verbs. There are a number of special-purpose verbs as well.

```
SORT            CUSTOMERS          WITH STATE = "IN"
verb            filename           selection expression

BY LAST-NAME    LAST-NAME  STREET  CITY  STATE
sort expression            output specifications

ID-SUPP         HEADING "INDIANA CUSTOMERS"      (P)
formatting modifier        formatting modifier   option
```

Example 1-6. ACCESS Query Syntax.
An ACCESS query is made up of different syntactical elements.

ACCESS verbs fall into six functional groups:

1. **Listing and Sorting Verbs.** These verbs perform basic listing and sorting operations on file items. They are described in Chapter 3.

LIST	LIST-ITEM
SORT	SORT-ITEM

2. **Select-List Verbs.** These verbs create and manipulate lists of item IDs from a database. They are described in Chapter 5.

SELECT	SAVE-LIST
SSELECT	GET-LIST
FORM-LIST	COPY-LIST
NSELECT	EDIT-LIST
QSELECT	DELETE-LIST

3. **Specialized Processing Verbs.** LIST-LABEL and SORT-LABEL create reports in special label format. COUNT, SUM, and STAT count, total, and average information in a database. REFORMAT and SREFORMAT also restructure file items. These verbs are described in Chapter 6.

LIST-LABEL	COUNT	REFORMAT
SORT-LABEL	SUM	SREFORMAT
	STAT	

4. **Tape Verbs.** These verbs copy items specified or selected by ACCESS queries to and from magnetic tape. They are described in Chapter 6.

   T-DUMP T-LOAD S-LOAD

5. **File Analysis Verbs.** These verbs display information about the use of file space and about changes in a file's contents. They are described in Chapter 6.

   CHECK-SUM FILE-TEST ISTAT
   HASH-TEST

6. **Forms Generation.** Forms generation allows you to position data on preprinted or system-generated forms. This function is described in Chapter 7.

   FORMS REPT
   SFORMS SREPT

In this chapter, you have seen many of the elements of an ACCESS query. You should be able to recognize the individual parts of a query by identifying the keyword that introduces it. In the next chapter we look more closely at the syntax of ACCESS queries, which will help you to write queries of your own.

CHAPTER 2

ACCESS Syntax

Chapter 2 introduces all the elements of ACCESS query syntax and describes the default behavior of the ACCESS processor. It explains how to enter different kinds of literal value (item IDs and other constants), how to enter queries that take up more than one line on the screen, and how to use certain syntax elements that make queries easier to use. Chapter 3 shows in more detail how to use these syntax elements to generate a variety of reports, from the simple to the more complex.

An ACCESS query can include all or some of the following parameters:

- A verb (required).
- A filename (required).
- A list of item IDs.
- Selection criteria.
- Sort expressions.
- Output specifications.
- Keywords that specify the report format.
- Options specific to the verb.

The verb and the filename are the only mandatory parameters in an ACCESS query; all other parameters are optional. The verb must be the first parameter in a command line. The remaining parameters are generally

entered in the order specified above, but need not be. Each ACCESS query is entered at the TCL prompt (>) and is executed by pressing the RETURN key.

The complete syntax for an ACCESS query is as follows:

verb [*file-modifiers*] *filename* [*items*] [*selection*] [*sort*] [*output*]
[*modifiers*] [(*options*)]

verb — specifies the operation to be performed on a database, such as displaying a list of item IDs or producing a report in a complex format. Every ACCESS query must begin with a verb.

file-modifiers — can be either DICT or ONLY. DICT specifies the file dictionary. If DICT is omitted, the data file is used. ONLY suppresses the default output specification and displays item IDs only.

filename — specifies a file accessible from your account. Every ACCESS query must include a filename.

items — can be either an explicit item list or a selection expression. An item list is one or more item IDs specifying items to be processed. Enclose each item ID in single quotes. A selection expression specifies one or more conditions an item ID must meet to be included in the report.

selection — specifies the conditions data in an item must meet for the item to be included in the report. Enclose literal values in double quotes.

sort — specifies by name one or more attributes which determine how items are to be sorted in the report. An ACCESS query can include multiple *sort expressions,* and items can be sorted in ascending or descending order.

output — specifies by name the attributes whose data will be listed in the report. You can also use a phrase for this parameter if your system supports user-defined phrases.

Print limiters can be used as output specifications to determine how multivalued attributes will be processed, which data from Controlling and Dependent attributes will be displayed, and which specific data elements will appear in the report.

modifiers are keywords that change the format of a report or specify special processing of the data (for example, calculating subtotals and grand totals). Modifiers immediately precede the name of the attribute they are to affect.

options any set of single-character options that modify the action of the verb. Precede the options you use with an open parenthesis; a closing parenthesis is optional.

By default, the results of an ACCESS query are displayed on your terminal screen. The report can be sent to the printer by using the LPTR modifier or the P option.

How Queries Are Processed

This section summarizes how the system recognizes and interprets the different parameters specified in an ACCESS query.

After you enter a query, the system identifies the verb and passes control to the ACCESS processor. The ACCESS processor identifies the parameters in a query by looking up in the following order:

1. The filename in the Master Dictionary.
2. All other words to see if they are defined in the Master Dictionary as connectives, modifiers, or relational operators.
3. The remaining unidentified words to see if they are defined in the file dictionary as attributes or phrases.
4. The remaining unidentified words to see if they are defined in the Master Dictionary as synonyms or phrases.

The processor treats all remaining words, or all words enclosed in double quotes or backslashes, as literal values. All words enclosed in single quotes are assumed to be item IDs.

If a word is not found in the file dictionary or the Master Dictionary, the following error message is displayed:

```
[24] THE WORD 'a' CANNOT BE IDENTIFIED
```

The ACCESS processor then generates the report by:

5. Applying *correlatives*. These convert data to an intermediate format used for further processing. Correlatives are fully explained in Chapter 8.
6. Processing selections, sorts, totals, and control breaks.
7. Processing *conversions*. These convert data from stored or intermediate format to the output format used in the report. Conversions are fully explained in Chapter 8.

Default ACCESS Processing

It is important to understand the default behavior of the ACCESS processor—in other words, what it normally does for you unless you specify otherwise.

Table 2-1 summarizes what occurs when any of the parameters in an ACCESS query are not explicitly specified. It also describes the relationships that the various parameters have to one another.

Table 2-1. ACCESS Query Parameters.

Parameter	Default Processing
filename	Unless the DICT modifier is included, the report is generated from the data file.
items	If there is no explicit item list, active select-list, or item selection expression, all items are processed. Processing means 1) included in the report, 2) compared against one or more selection expressions, and 3) sorted.

	If there is an explicit item list, the items are processed in the order specified. Explicit item lists and item selection expressions take precedence over an active select-list.
selection	If there is no explicit item list or active select-list, all items are compared against the selection expressions.
sort	If no sort expression is specified and if the verb has sorting capabilities, the items are displayed in ascending order by item ID. If multiple sort expressions are specified, they are processed from left to right.
output	If no output specification is included in the query, the report for a data file consists of the default output specifications defined by a sequence of numeric item IDs. User-defined @ phrases can also specify default output specifications if your system supports them. If no *print limiters* are specified, all values from multivalued attributes as well as all data from other attributes are included in the report.
modifiers	If no modifiers are included in the query, the report uses: • no control breaks. • a single space between items. • headings that consist of the page number, time, date, and an end-of-list message. • no footings. • column headings that are repeated on every page. • item IDs. • the terminal screen as the destination. • the default output specifications. • paging to halt terminal output at the end of each page. • data from the file on disk.
options	Defaults for options are specific to the verb.

Defining Default Output Specifications

When creating a file, you will usually identify the attributes whose contents are to be displayed in reports. It is possible, however, to create *default*

output specifications in the dictionary of the referenced file. These will take effect only if explicit output specifications are *not* included in an ACCESS statement. For example, assume that the default output specifications for the CUSTOMERS file are defined to include the data for FIRST-NAME, LAST-NAME, CITY, and STATE. With the new default in effect, entering:

```
>LIST CUSTOMERS
```

is equivalent to entering:

```
>LIST CUSTOMERS FIRST-NAME LAST-NAME CITY STATE
```

The most common way to define default output specifications is to create a sequence of numbered Attribute Definition items.

If the file dictionary contains Attribute Definition items with numeric item IDs, and if they form a consecutive numeric sequence starting with 1, these items are used as default output attributes. Each item in the sequence must contain an A code (or S) in line 1.

If an Attribute Definition item in a numeric sequence contains an X instead of an A in line 1, that attribute will *not* be displayed as one of the default output attributes. However, the attributes defined by items that follow this X item in the numeric sequence *will* be included in the report. The X item allows you to preserve the sequence of numeric Attribute Definition items.

One way to create a numeric sequence of default output attributes is to copy an existing set of Attribute Definition items with the COPY verb. For example, if the CUSTOMERS dictionary has Attribute Definition items for FIRST-NAME (Attribute 1), LAST-NAME (Attribute 2), CITY (Attribute 3), and STATE (Attribute 4), you can copy them, giving them the item IDs "1", "2", "3", and "4". Once this is done, when you enter:

```
>LIST CUSTOMERS
```

the data from all four attributes will be displayed by default.

If you want to exclude Attribute 3, CITY, from the sequence of default output attributes, change the definition code in line 1 of the Attribute Definition item "3" from "A" to "X". Now only Attributes 1, 2, and 4 will be displayed by default.

Creating an @LPTR Phrase

Some systems allow you to create special dictionary items known as *phrases*. These phrases can be used to define default output specifications for either the printer or the terminal screen.

A phrase can include any parameters from an ACCESS query except for a verb or a filename. The use of phrases is described at the end of this chapter, in the section, "Using Phrases."

Creating a phrase in the CUSTOMERS dictionary whose item ID is @LPTR defines the default output specifications for reports sent to the printer. Once an @LPTR phrase is defined, any ACCESS statement that contains the LPTR modifier or its equivalent, the P option, and that does not include an output specification, will print data from the attributes named in the @LPTR phrase. For example, the following @LPTR phrase specifies customers' first and last names, city, and state:

```
        @LPTR
001     I
002     FIRST-NAME LAST-NAME
003     CITY STATE
```

A definition code of "I" defines this file item as a phrase. When the following query is entered:

```
>LIST CUSTOMERS LPTR
```

the results will be the same as:

```
>LIST CUSTOMERS FIRST-NAME LAST-NAME CITY STATE
  LPTR
```

Creating an @ Phrase

Creating a phrase in the CUSTOMERS dictionary whose item ID is @ defines the default output specifications for reports. If there is also an @LPTR phrase in the dictionary, the output specifications in the @ phrase will be used only for reports sent to the terminal screen; the @LPTR phrase determines what is output to the printer. If there is no @LPTR phrase in the dictionary, the @ phrase is used for reports sent either to the screen or to the printer.

For example, you might define the following @ phrase in the CUSTOMERS dictionary:

```
     @
001  I
002  FIRST-NAME LAST-NAME
003  CITY STATE
```

In this case, you can produce the default report by entering:

```
>LIST CUSTOMERS
```

Example 2-1 shows the resulting report.

```
PAGE   1                                          10:33:54  01 NOV 1989

CUSTOMERS... First Name Last Name. City........ State

HJENK7129    HAROLD      JENKINS     INDIANAPOLIS    IN
JBOHA5422    JOHN        BOHANNON    BOSTON          MA
JBROW6749    JAMES       BROWN       BOSTON          MA
JBUCK6488    JULIE       BUCKLER     LINCOLN         IN
BLEAR6803    BILL        LEARY       BOSTON          MA
JMASO6378    JULIA       MASON       LINCOLN         IN
AORLA5993    AMY         ORLANDO     VENICE          CA
SPIRS5289    SANDRA      PIRS        WINSTON         NC
MASHX5777    MARY        ASH         INDIANAPOLIS    IN
AEDWA5224    ANTHONY     EDWARDS     SUDBURY         MA
JPEER5993    JAN         PEERCE      LOS ALTOS       CA
RPIER5539    RICK        PIERCE      RIDGEWOOD       NJ
AJOHN5396    ANNE        JOHNSON     LOUISVILLE      KY
HJOHN7265    HENRY       JOHNSON     OMAHA           NB
HHIGG6849    HENRY       HIGGINS     OMAHA           NB
DEDGE6635    DAVID       EDGECOMB    MIAMI           FL
AMEAD5619    ANDREW      MEADE       SUDBURY         MA
BLAMP6196    BOB         LAMPSON     BOSTON          MA

18 ITEMS LISTED.
```

Example 2-1.
Using an @ Phrase to specify default output.

Suppressing Default Output Specifications

The default output specifications can be suppressed by including the ONLY modifier in the ACCESS query. In this case, the report includes item IDs

only. For example, the following query limits the output from the CUSTOMERS dictionary to item IDs:

```
>LIST ONLY DICT CUSTOMERS
```

Example 2-2 shows a page from the resulting report.

```
PAGE   1                                          10:52:45  01 NOV 1989

CUSTOMERS

ROAD
FIRST-NAME
REPT
CITY-ST
PHONE
LAST-NAME
ZIP
STREET
LATE-NOTICE
STATE
SAVE@
CUST-ID
CITY
CUSTOMERS
LABEL
SOC-SEC
GROUP
CUSTOMERS
EXTENSION
```

Example 2-2.
The file modifier ONLY produces a report that lists item IDs.

Entering Literal Values

There are two types of literal value that can be included in an ACCESS query: item IDs and constants. It is recommended that you enclose item IDs in single quotes. Single quotes identify the element as an item ID, no matter where in the ACCESS query it occurs.

For example:

```
>LIST CUSTOMERS NAME '1' '2' '3'
```

lists items with customer ID numbers 1, 2, and 3, even though the item IDs do not immediately follow the filename.

The next example:

```
>LIST CUSTOMERS NAME "1" "2" "3"
```

produces unexpected results because the numbers enclosed in double quotes are not recognized as item IDs.

The way an item ID is specified in a query can affect the amount of processing required to retrieve the data. For example, the following query uses the item ID AORLA5593 to locate the group of frames where the item is stored, then retrieves the item.

```
>LIST CUSTOMERS 'AORLA5593'
```

This works much more quickly than the query which follows, which uses the selection expression "= 'AORLA5593' ". When this query is processed, every item ID in the CUSTOMERS file is compared to the literal AORLA5593 until it is found.

```
>LIST CUSTOMERS = 'AORLA5593'
```

The second method is probably most useful when you are looking for item IDs that contain a particular substring, as in the following example:

```
>LIST CUSTOMERS = 'AORLA]'
```

String searching is explained more fully in Chapter 3.

Constants are used in selection expressions, print limiting expressions, and as arguments to certain modifiers. They identify specific dates, times, dollar amounts, customers' last names, etc. Enclose each constant in double quotes or backslashes (\ ).

The following query lists all customers whose last name is Buckler:

```
>LIST CUSTOMERS WITH LAST-NAME "BUCKLER"
```

If you do not enclose the literal BUCKLER in double quotes, ACCESS treats it as if it were either an item ID or an output specification, and returns the following message:

```
[ 24 ] THE WORD "BUCKLER" CANNOT BE IDENTIFIED
```

Entering Multiple-Line Queries

ACCESS queries are sometimes longer than 80 characters and therefore might not fit on a single line of the terminal screen. You can, however, keep typing a command line, even though it wraps onto the next line. To make a long command line easier to read on the screen, you can break the line and continue typing on the next line. To break a line, press CTRL-_ (underscore), and then press the RETURN key. A colon prompt appears, and you can continue to type the query.

For example:

```
>LIST CUSTOMERS WITH LAST-NAME > "MARTIN" AND _
 :WITH STATE EQ "MI" LAST-NAME FIRST-NAME CITY _
 :STATE ZIP
```

Blank spaces are included at the ends of each line to keep AND separate from WITH, and CITY separate from STATE. A colon (:) is displayed at the beginning of each continuation line.

Using Throwaway Connectives

The Pick system includes a number of keywords that can be used anywhere in a query to make it more like a natural-language sentence. These keywords are called *throwaway connectives*. Throwaway connectives do not change the meaning of the ACCESS query and they do not affect the appearance or content of the resulting report.

Table 2-2 lists some of the system-supplied throwaway connectives.

Table 2-2. System-Supplied Throwaway Connectives.

A	DATA	ITEMS
AN	FILE	OF
ANY	FOR	OR
ARE	IN	THE

You can, of course, add your own throwaway connectives to the Master Dictionary. Simply copy any throwaway connective to the name of a new throwaway. For example:

```
>COPY MD A
TO :DISPLAY
```

creates a new throwaway connective, DISPLAY, that you might use in an ACCESS query to introduce a list of output specifications.

The following ACCESS queries demonstrate the use of throwaway connectives:

```
>LIST ANY CUSTOMERS WITH LAST-NAME = "JOHNSON"
>LIST THE ORDERS FILE
>SORT CUSTOMERS DATA BY CITY
```

Using Phrases

As was mentioned earlier, some systems (such as uniVerse, Mentor, and Prime INFORMATION) support user-defined phrases in dictionary items.

A phrase can be made up of any elements of an ACCESS query except for a verb, a filename, or a parenthetical option. Here are some sample phrases:

```
WITH AMOUNT > "100"
BY AMOUNT BY QTY
AMOUNT QTY DATE
HEADING "LARGE ORDERS"
WITH AMOUNT > "100" BY AMOUNT BY QTY AMOUNT QTY DATE
   HEADING "LARGE ORDERS"
```

Phrases are especially useful for saving complex or lengthy report formatting specifications. You can save and reuse phrases by defining them as items in the file dictionary. For example, you might define the last phrase in the preceding list as an item in the ORDERS dictionary and call it LARGE-ORDERS. You could then produce a customized report at any time by entering:

```
>SORT ORDERS LARGE-ORDERS
```

The preceding query produces a report that contains a list of the amounts, numbers, and dates of all orders larger than $100, sorted in ascending order

by amount, then by number. Each page of the report begins with the heading "LARGE ORDERS".

When creating a phrase, there is no need to break the lines in any particular way, except for readability. Nor is it necessary to use CTRL-_ (underscore) to break the lines. The dictionary item for LARGE-ORDERS might look like this:

```
        LARGE-ORDERS
001     I
002     WITH AMOUNT > "100"
003     BY AMOUNT BY QTY
004     AMOUNT QTY DATE
005     HEADING "LARGE ORDERS"
```

The I code in line 1 is required to identify the item as a phrase.

The following rules apply when creating phrases:

- Carriage returns that are used to break phrases into multiple lines are automatically converted to blank spaces except when text is enclosed in quotes or backslashes (\ ). This might occur, for example, if a phrase includes header text extending over two or more lines. Also, if a line begins with a colon, the carriage return at the end of the preceding line is not converted to a space.
- Any line that begins with an exclamation point is treated as a comment and ignored during processing. If you put an exclamation point at the beginning of line 1 (that is, before the I code), the entire phrase will be ignored.
- A null phrase (i.e., a phrase containing no words apart from comments) is ignored.

One or more phrase names can themselves be included in a phrase definition. This allows a phrase to "call" and expand other phrases.

Avoid creating loops! Do not include a phrase name that calls another phrase that refers back to itself.

Phrases can also be defined in the Master Dictionary. If user-defined phrases are located in the Master Dictionary, you can include them in any ACCESS query you enter. For example, you could create a phrase called CALCULATIONS that displays a listing of only those dictionary items

containing an A or F code in lines 7 or 8. This phrase could be used when querying any file dictionary you have access to.

ACCESS Verbs and Keywords

The following two sections contain summaries of all ACCESS verbs and keywords that are contained in the Master Dictionary. All SMA standard verbs and keywords are included, as well as several others that are available on some systems but not on others. Non-SMA standard verbs and keywords are marked with an asterisk.

ACCESS Verbs

Table 2-3 summarizes the function of each ACCESS verb.

Table 2-3. ACCESS Verbs.

Verb	Description
CHECK-SUM	Provides statistical information about items and files.
COPY-LIST	Copies a previously saved select-list to the terminal, a printer, another saved select-list, or a file.
COUNT	Counts the number of items in a file that meet the specified criteria.
DELETE-LIST	Deletes a previously saved select-list.
EDIT-LIST	Invokes the Editor to change the contents of a saved select-list.
FILE-TEST	Provides file hashing statistics that can be used to evaluate the efficiency of file storage. FILE-TEST includes the functions of both the ISTAT and HASH-TEST verbs, which it supersedes.
FORM-LIST	Creates a select-list that consists of the data in a file rather than its item IDs.

Verb	Description
*FORMS	Prints items on forms. Not supported on all systems.
GET-LIST	Retrieves a previously saved select-list for processing by a subsequent command.
*HASH-TEST	Provides information about how file items hash into groups specified by a given modulo. This helps you select the best modulo for a file when reallocating disk space.
*ISTAT	Provides file hashing statistics that can be used to evaluate the efficiency of file storage.
LIST	Lists the items in a file that meet the specified criteria. Many different types of report can be generated using the modifiers and options available with LIST.
LIST-ITEM	Displays items, listing data for each attribute.
LIST-LABEL	Prints items in a label format.
*NSELECT	Creates a select-list of items found in an active select-list but not found in the specified file. Not supported on all systems.
*QSELECT	A synonym for FORM-LIST.
REFORMAT	Creates a second file or tape using output from an ACCESS report.
*REPT	Prints multiple items on forms. Not supported on all systems.
S-DUMP	Copies items that meet the specified criteria to magnetic tape in sorted order.
SAVE-LIST	Saves a select-list under a specified name.
SELECT	Creates a select-list of item IDs. This list of items can then be submitted for further processing by other verbs.
*SFORMS	Prints items in sorted order on forms. Not supported on all systems.
SORT	Lists the items in a file that meet the specified criteria in sorted order. The SORT verb provides the same flexibility in generating reports as the LIST verb.
SORT-ITEM	Displays a sorted list of items, listing data for each attribute. See LIST-ITEM.

Verb	Description
SORT-LABEL	Creates and sorts labels from items that meet the specified criteria. See LIST-LABEL.
SREFORMAT	Creates a second file or tape in sorted order using output from an ACCESS report. See REFORMAT.
*SREPT	Prints multiple items in sorted order on forms. Not supported on all systems.
SSELECT	Creates a select-list that consists of sorted item IDs that meet the specified criteria. This list of items can then be submitted for further processing by other verbs. See SELECT.
STAT	Provides a count, average, and total of the numeric data for a specified attribute.
SUM	Totals the numeric data for a specified attribute.
T-DUMP	Saves items that meet the specified criteria to magnetic tape.
T-LOAD	Restores items that meet the specified criteria from magnetic tape.

ACCESS Keywords

There are many keywords (also called connectives or modifiers) that can be included to expand ACCESS queries. A few have already been introduced in Chapter 1 (for example, BY, HEADING, ID-SUPP, and WITH).

Keywords can be used to:

- Specify a comparison between attributes or between an attribute and a constant using relational operators such as GREATER THAN, EQUAL TO, and LESS THAN.
- Sort items in ascending or descending order according to data in one or more attributes.
- Format reports according to your own specifications for headings, footings, line spacing, and more.
- Retrieve and display items from magnetic tape.

- Print items on the line printer.
- Make a query more like a natural-language sentence.

In order to use a keyword in an ACCESS query, there must be a corresponding entry for that keyword in the Master Dictionary. Table 2-4 lists the ACCESS keywords. Some keywords have an equivalent parenthetical option.

Table 2-4. ACCESS Keywords.

Keyword	Option	Description
A, AN	—	Makes a query more English-like.
AFTER	—	Greater Than.
AND	—	Joins selection expressions.
ANY	—	Makes a query more English-like.
ARE	—	Makes a query more English-like.
BEFORE	—	Less Than.
BREAK-ON	—	Specifies breakpoints in a report.
BY	—	Sorts the specified attribute in ascending order.
BY-DSND	—	Sorts the specified attribute in descending order.
BY-EXP	—	Separates items with multivalued attributes into multiple items and sorts in ascending order.
BY-EXP-DSND	—	Separates items with multivalued attributes into multiple items and sorts in descending order.
COL-HDR-SUPP	(C)	Suppresses time and date heading, column headings, and end-of-list message.
DATA	—	Makes a query more English-like.
DBL-SPC	—	Double-spaces items in the report.
DET-SUPP	(D)	Suppresses detail output when used with TOTAL or BREAK-ON modifiers.
DICT	—	Specifies the file dictionary.

Keyword	Option	Description
EACH	—	Selects an item only if each value in a multivalued attribute meets the specified condition. Used with WITH connective.
*END-WINDOW	—	Terminates a WINDOW phrase (see WINDOW).
EQ	—	Equal To.
EVERY	—	Selects an item only if every value in a multivalued attribute meets the specified condition. Used with WITH connective.
—	(F)	Starts a new page for each item when used with the LIST-ITEM, and SORT-ITEM verbs.
FILE	—	Makes a query more English-like.
FOOTING	—	Defines footer for report page.
FOR	—	Makes a query more English-like.
GE	—	Greater Than or Equal To.
GRAND-TOTAL	—	Prints the specified text on the grand total line.
GT	—	Greater Than.
HDR-SUPP	(H)	Suppresses default heading.
HEADING	—	Overrides default heading.
ID-SUPP	(I)	Suppresses item ID display.
IF	—	Begins a selection expression.
IN	—	Makes a query more English-like.
ITEMS	—	Makes a query more English-like.
LE	—	Less Than or Equal To.
*LIKE	—	Retrieves data that is phonetically similar to the specified criteria. On some systems this function is performed by the SAID or SPOKEN keywords.
LPTR	(P)	Sends output to printer.
*MATCHING	—	Finds data that matches a specified constant or string.
LT	—	Less Than.

Keyword	Option	Description
NE	—	Not Equal To.
NO	—	Not Equal To.
*NOT.MATCHING	—	Finds data that does not match a specified constant or string.
NOPAGE	(N)	Suppresses page pause.
NOT	—	Not Equal To.
OF	—	Makes a query more English-like.
ONLY	—	Suppresses the default output specification and displays only item IDs.
OR	—	Joins selection expressions.
*SAID	—	Retrieves data that is phonetically similar to the specified criteria. See also the LIKE keyword.
*SPOKEN	—	Retrieves data that is phonetically similar to the specified criteria. See also the LIKE keyword.
SUPP	(H)	Suppresses default heading.
TAPE	—	Obtains data from magnetic tape.
THE	—	Makes a query more English-like.
TOTAL	—	Calculates and displays totals for the data in an attribute.
USING	—	Specifies the file to be used to interpret a file.
*WINDOW	—	Defines a window for printing multivalues on forms.
WITH	—	Begins a selection expression.
*WITHIN	—	Retrieves and lists subitems of a specified item.
WITHOUT	—	Equivalent to WITH NO and WITH NOT.
—	(Y)	Prints translation of A-correlative DICT entries.

CHAPTER 3

Producing Reports with LIST and SORT

This chapter uses two multipurpose ACCESS verbs, LIST and SORT, to illustrate the full range of reports you can generate. Specifically, this chapter covers:

- Selection expressions.
- Sort expressions.
- Which attributes to display or print in the report (output specifications).
- Which multivalues to display or print in the report (print limiters).

In addition, this chapter discusses some special connectives that can be included in ACCESS queries.

The LIST and SORT Verbs

LIST and SORT are two general-purpose ACCESS verbs. LIST displays or prints items contained in either a dictionary or a data file and gives you great flexibility in generating reports from these entries. SORT has all the

capabilities of LIST but includes the added ability to list items in sorted order.

The LIST verb can specify:

1. Which items to select for inclusion in the report. This is done by explicitly specifying items by their item IDs or by using a selection expression to select only those items that match the specified criteria. These two approaches can also be combined to specify a subset of items and then test them against selection criteria.
2. How to sort the items. Items can be sorted in ascending or descending order, based on data referenced by any attribute name.
3. Which attributes to display in the report. An item might have 10 or 12 attributes, but a report might only list the contents of two of them. By default, LIST displays no attributes, only item IDs.
4. How multivalued attributes are treated. A report can display all values from multivalued attributes or only those values that meet the specified condition.

You can also define report headings, footings, and vertical spacing, calculate subtotals and totals for the data in various attributes, and more. These formatting parameters are described in Chapter 4.

Using Selection Expressions

In Chapter 2 we learned how to select items to include in a report by specifying an item list in the query. In an item list each item is explicitly specified.

You can also select items for inclusion in a report by using a *selection expression.*

Selecting Items by Item ID

There are two kinds of selection expression. One kind is used to select items by specifying the conditions an *item ID* must meet in order for the item to be

included in the report. That's the kind of selection expression we'll look at in this section. The other kind selects items by specifying what conditions *data* in the file must meet in order for the item to be included; this is discussed in the following section. A selection expression that is used to select item IDs compares each item-ID in the file to the criteria specified in the expression.

You can use relational operators (such as Less Than) to compare the item IDs to a constant. You can also use pattern matching.

In the ORDERS file, for example, sequentially issued order numbers are used as item IDs. To list just the order numbers higher than 10110, enter:

```
>LIST ORDERS > "10110"
```

The preceding example uses the relational operator " > " (Greater Than) to select only those item IDs with numbers above 10110.

Selecting Items by Data Attribute

A selection expression can also be used to specify the conditions that *data* in an item must meet for the item to be included in a report. Selection expressions test the data in one or more attributes by comparing it against the specified criteria.

Selection criteria can be specified as follows:

- Use relational operators (such as Less Than) to compare the data in an attribute to a constant.
- Use pattern matching to specify a character string that must appear at the beginning, middle, or end of an attribute's data.

Selection expressions have the following syntax:

WITH [**EACH**] [*attribute-name*] [*rel-op*] *value-list* [[**AND** | **OR**]
WITH [**EACH**] [*attribute-name*] [*rel-op*] *value-list*] ...

EACH — specifies that all values in a multivalued attribute must meet the specified condition if the item is to be listed.

attribute-name — is the attribute whose data is compared to the specified criteria. If no attribute is specified,

ACCESS compares the item ID to the specified criteria.

rel-op is a relational operator such as Less Than (<). For a complete list of valid relational operators, see Table 3-1 later in this chapter.

value-list is one or more constants, enclosed in double quotes.

You can use the AND and OR connectives to create a compound selection expression by 1) specifying more than one selection expression for a single attribute, or 2) defining one or more selection expressions for additional attributes. When you use AND or OR in a compound selection expression, you must repeat the keyword WITH.*

For example, the following query displays the item IDs of customers (items) from the CUSTOMERS file whose last name is Andrews:

```
>LIST CUSTOMERS WITH LAST-NAME "ANDREWS"
```

The next query lists customers whose last name is Andrews and who live in the city of Newton:

```
>LIST CUSTOMERS WITH LAST-NAME "ANDREWS" AND
  WITH CITY "NEWTON"
```

The next query uses the OR connective to list all customers whose last name is Andrews (and who live anywhere) as well as all customers who live in Newton (no matter what their last name is).

```
>LIST CUSTOMERS WITH LAST-NAME "ANDREWS" OR WITH
  CITY "NEWTON"
```

The OR connective is optional. If multiple selection expressions are included in a query and the logical operators AND and OR are omitted, OR is assumed. Thus the preceding query could just as well be entered as follows:

---

* On some systems it is not necessary to repeat the WITH connective after AND or OR. On uniVerse and Prime INFORMATION systems, for example, you can enter the preceding command like this:

```
>LIST CUSTOMERS WITH LAST-NAME "ANDREWS" AND CITY
    "NEWTON"
```

Try entering compound selection expressions both ways on your system to see which syntax is supported.

```
>LIST CUSTOMERS WITH LAST-NAME "ANDREWS" WITH
   CITY "NEWTON"
```

The following sections contain detailed information about forming selection expressions.

Using Relational Operators

Relational operators can be used in selection expressions to compare data to a constant. If the data is left-justified, as defined by line 9 of the Attribute Definition item, the data is compared from left to right. Any characters that are not identical are converted to their numerical ASCII equivalents and compared. Higher ASCII equivalents are considered "greater."

If the data is right-justified, first a numerical comparison is made. Nonnumerical characters are then converted to their numerical ASCII equivalents and compared.

Table 3-1 lists the relational operators that can be used in ACCESS queries. There are at least two ways to specify each operator.

Table 3-1. Relational Operators.

Operator	Function
= or EQ	Equal to.
# or NE or NO [T]	Not equal to or null.
> or GT or AFTER	Greater than.
< or LT or BEFORE	Less than.
>= or GE	Greater than or equal to.
<= or LE	Less than or equal to.

The following query selects customers whose zip code is a number less than 21300:

```
>LIST CUSTOMERS WITH ZIP < "21300"
```

Relational operators can be used with both alphabetic and numeric characters. For example, the following query selects customers whose last names alphabetically precede Mansfield:

```
>LIST CUSTOMERS WITH LAST-NAME < "MANSFIELD"
```

If no relational operator is explicitly specified, "Equal To" is assumed. For example:

```
>LIST CUSTOMERS WITH CITY "BALTIMORE"
```

If no attribute is specified, the item ID is assumed. For example:

```
>LIST ORDERS WITH < "10183"
```

The preceding query lists all orders whose item IDs are less than 10183. This is, in fact, the same as selecting specified item IDs, as described earlier in this chapter. Normally in such instances the WITH connective is omitted.

Using Logical Connectives

There are two logical connectives that can be used in ACCESS queries to form compound selection expressions: AND and OR. These connectives correspond to the Boolean operators by the same names. Logical connectives are useful for defining complex selection expressions.

The AND connective specifies that *all* of the specified criteria must be met if an item is to be included in the report. For example, the following query includes three different selection expressions:

```
>LIST ORDERS WITH DATE "11/11/87" AND WITH
   TOTAL.AMT > "100" AND WITH CUST# > "330"
```

All three of the following conditions must be met for each order item to be selected:

1. An order date of November 11, 1987.
2. An order amount of more than $100.00.
3. A customer number higher than 330.

The report will contain the item IDs of only those items in the ORDERS file that meet all three conditions. An ACCESS query can include up to nine AND phrases.

The OR connective specifies that *at least one* of the specified criteria must be met if an item is to be included in the report. For example:

```
>LIST ORDERS WITH DATE "11/11/87" OR WITH
   TOTAL.AMT > "100" OR WITH CUST# > "330"
```

The preceding query produces a report quite different from the one that uses the AND connective. It selects *all* of the following items:

1. All orders taken on November 11, 1987.
2. All orders taken on any date for an amount greater than $100.00.
3. All orders taken for all customers whose ID numbers are greater than 330.

If no logical connective is used in a compound selection expression, the system assumes OR. An ACCESS query can include any number of OR phrases.

If more than one logical connective is used in a query, the selection expressions are evaluated from left to right. For example, let's assume you want to display the item for a certain customer whose last name is SMITH. You think the customer lives in Massachusetts or Maine, but you aren't sure. You also think the name of the town is Rockport. You could enter:

```
>LIST CUSTOMERS WITH LAST-NAME "SMITH" AND WITH
   CITY "ROCKPORT" AND WITH STATE "MA" OR "ME"
```

This query generates a list of all customers whose last name is SMITH, who live in a city called Rockport, and who live in Massachusetts or Maine.

String Searching

Instead of using relational operators and constants, you can specify alphanumeric character strings as part of a selection expression in an ACCESS query. The data for the specified attribute must then contain the character string if an item is to be included in the report.

For example, the following query displays the last and first names of customers whose last names begin with the letter J.

```
>LIST CUSTOMERS WITH LAST-NAME = "J]" LAST-NAME
   FIRST-NAME
```

Example 3-1 shows the resulting report.

Soundex algorithm differently, and all systems do not support it. We mention two implementations here. The first, used by ADDS Mentor, uses the keyword LIKE. The second, used by Prime INFORMATION and uniVerse, uses the keywords SAID or SPOKEN, which are synonymous.

The LIKE* connective compares the data in a specified attribute to an alphabetic string. Unlike the string searching feature described in the preceding section, however, the LIKE connective requires the data to be merely phonetically *similar* to the specified string. This type of selection expression is particularly useful if you aren't quite sure how the data might be spelled.

For example, the following query searches for customers whose last name is something like PIERCE.

```
>LIST CUSTOMERS WITH LAST-NAME LIKE "PIERCE"
  LAST-NAME  FIRST-NAME
```

Example 3-2 shows a report that lists three customers: Pirs, Peerce, and Pierce.

```
PAGE   1                                          10:56:57  01 NOV 1989

CUSTOMERS... Last Name. First Name

SPIRS5289    PIRS        SANDRA
JPEER5993    PEERCE      JAN
RPIER5539    PIERCE      RICK

3 ITEMS LISTED.

>
```

Example 3-2.
The LIKE connective lets you find data that "sounds like" the specified string.

* On Prime INFORMATION and uniVerse systems, the LIKE connective is synonymous with the MATCHING connective, used for string searching.

Only one alphabetic character string can be specified after the LIKE connective in an ACCESS query.

Using SORT Expressions

The SORT verb and its variations, including SORT-ITEM and SORT-LABEL, produce reports in which file items are displayed or printed:

1. In order according to the data specified by any attribute name.
2. In ascending or descending order.

As discussed earlier, LIST generates unsequenced reports, that is, reports in which the items appear in no particular order. It is extremely useful to be able to list file items in sorted order. This makes it possible, for example, to list items in order by customer name, by zip code, or by state. You can also sort by numeric values such as dollar figures and dates.

To list file items in ascending order, include the word BY followed by the name of the desired attribute. To list items in descending order, use the BY-DSND modifier.

The following query lists customers in ascending order by last name (beginning with A):

```
>SORT CUSTOMERS BY LAST-NAME
```

The following query lists customers in descending order by state (beginning with Z):

```
>SORT CUSTOMERS BY-DSND STATE
```

The justification (line 9 in the Attribute Definition item) determines the type of sort: if the attribute is left-justified (L, T, or U), the sort is alphabetical, whereas if the attribute is right-justified (R), the sort is numerical if the data is numerical; nonnumerical characters are sorted according to their ASCII value.

Figure 3-1 shows the same set of item IDs sorted in ascending order. The first list, however, is left-justified; the second is right-justified. Notice the difference in the sort order.

Left-Justified	Right-Justified
1	1
10	2
100	10
101	11
10101	20
11	22
110	100
111	101
2	110
20	111
200	200
201	201
22	220
220	222
222	10101

Figure 3-1. Sorting Numbers.
The column of left-justified numbers is not sorted numerically, but rather from left to right. Right-justified numbers are sorted in numerical order.

Sorting By Multiple Attributes

A single ACCESS query can specify that items be sorted according to the data in several attributes. This allows you to specify primary and secondary sort keys. Multiple sort expressions are processed from left to right, i.e., in the order in which they appear in the ACCESS query. For example:

```
>SORT CUSTOMERS BY STATE BY LAST-NAME BY
  FIRST-NAME STATE LAST-NAME FIRST-NAME
```

The report in Example 3-3 displays all items in the CUSTOMERS file sorted by state. Within each state grouping, items are sorted by the customers' last names, and within each last name grouping, items are sorted by first name.

```
PAGE   1                                              10:57:31  01 NOV 1989

CUSTOMERS... State Last Name. First Name

AORLA5993        CA ORLANDO     AMY
JPEER5993        CA PEERCE      JAN
DEDGE6635        FL EDGECOMB    DAVID
MASHX5777        IN ASH         MARY
JBUCK6488        IN BUCKLER     JULIE
HJENK7129        IN JENKINS     HAROLD
JMASO6378        IN MASON       JULIA
AJOHN5396        KY JOHNSON     ANNE
JBOHA5422        MA BOHANNON    JOHN
JBROW6749        MA BROWN       JAMES
AEDWA5224        MA EDWARDS     ANTHONY
BLAMP6196        MA LAMPSON     BOB
BLEAR6803        MA LEARY       BILL
AMEAD5619        MA MEADE       ANDREW
HHIGG6849        NB HIGGINS     HENRY
HJOHN7265        NB JOHNSON     HENRY
SPIRS5289        NC PIRS        SANDRA
RPIER5539        NJ PIERCE      RICK

18 ITEMS LISTED.
```

Example 3-3.
The report is sorted first by STATE, then by LAST-NAME, then by FIRST-NAME.

Sorting Data in Multivalued Attributes

The BY-EXP and BY-EXP-DSND modifiers sort the data in multivalued attributes. These modifiers make it possible to produce sorted reports from files whose items contain more than one value per attribute.

For example, the items in the ORDERS file contain the following multivalued attributes:

- TITLE
- SHORT.TITLE
- BOOKCODE
- NUMBER
- PRICE
- LINE.AMT

This is because one order can include more than one title. The BY-EXP and BY-EXP-DSND modifiers ensure that all data from an order item is included in a report.

The query shown in Example 3-4 produces a report that sorts items by short title and includes data from four multivalued attributes. The report breaks after each short title.

```
>SORT ORDERS BY-EXP SHORT.TITLE BREAK-ON
  SHORT.TITLE BOOKCODE QTY LINE.AMT
```

```
PAGE   1                                          10:58:01  01 NOV 1989

ORDERS Short Title Book Code Qty Subtotal

 10101 DATABASE    N01         2   $19.90
 10102 DATABASE    N01         1    $9.95
 10103 DATABASE    N01         2   $19.90
 10104 DATABASE    N01         3   $29.85
 10105 DATABASE    N01         1    $9.95
 10107 DATABASE    N01         9   $89.55
 10110 DATABASE    N01        12  $119.40
 10114 DATABASE    N01         5   $49.75
 10115 DATABASE    N01         1    $9.95
 10118 DATABASE    N01         3   $29.85

       ***

 10102 OPERATING   N02         3   $56.25
 10104 OPERATING   N02         5   $93.75
 10106 OPERATING   N02         3   $56.25
 10107 OPERATING   N02        34  $637.50
 10108 OPERATING   N02        10  $187.50
 10110 OPERATING   N02         3   $56.25
```

Example 3-4.
The report is sorted by the multivalues in the attribute SHORT.TITLE.

Each short title appears in the report as a separate item. Notice, for example, that there are two separate entries for order item 10110. This means that the customer ordered at least two different books.

Displaying Selected Attributes

Part of designing a report is specifying the data to be output. For example, let's assume you want to generate a report from the ORDERS file such as the one shown in Example 3-5.

```
PAGE   1                                          10:54:53  01 NOV 1989

CUSTOMERS... Last Name. First Name

HJENK7129    JENKINS     HAROLD
JBOHA5422    BOHANNON    JOHN
JBROW6749    BROWN       JAMES
JBUCK6488    BUCKLER     JULIE
BLEAR6803    LEARY       BILL
JMASO6378    MASON       JULIA
AORLA5993    ORLANDO     AMY
SPIRS5289    PIRS        SANDRA
MASHX5777    ASH         MARY
AEDWA5224    EDWARDS     ANTHONY
JPEER5993    PEERCE      JAN
RPIER5539    PIERCE      RICK
AJOHN5396    JOHNSON     ANNE
HJOHN7265    JOHNSON     HENRY
HHIGG6849    HIGGINS     HENRY
DEDGE6635    EDGECOMB    DAVID
BLAMP6196    LAMPSON     BOB
AMEAD5619    MEADE       ANDREW

18 ITEMS LISTED.
```

Example 3-5.
Output specifications let you list specific attributes in a report.

The ORDERS dictionary defines twelve different attributes, but the data in only two of them is pertinent to this report. To produce the report shown, you would include the names of those two attributes in the query:

```
>LIST CUSTOMERS LAST-NAME FIRST-NAME
```

This explicit list of attributes is called an *output specification*. The report also includes item IDs, which are always displayed unless they are explicitly suppressed with the ONLY modifier.

If an output specification is not included in a query, the system produces a default display. The section, "Default ACCESS Processing," in Chapter 2, describes this fully.

The report shown in Example 3-5 displays the data in columnar format. This is the default format for ACCESS reports, unless the sum of the column widths exceeds the page width as defined by the TERM verb. In this case, the report lists the data vertically, in noncolumnar format.*

For example, had the columns been too wide for the screen, items in the preceding report would be displayed as in Example 3-6.

```
PAGE   1                                          10:37:11  01 NOV 1989


CUSTOMERS : HJENK7129
Last Name   JENKINS
First Name   HAROLD

CUSTOMERS : JBOHA5422
Last Name   BOHANNON
First Name   JOHN

CUSTOMERS : JBROW6749
Last Name   BROWN
First Name   JAMES

CUSTOMERS : JBUCK6488
Last Name   BUCKLER
First Name   JULIE

CUSTOMERS : BLEAR6803
Last Name   LEARY
First Name   BILL
```

Example 3-6.
If the sum of the column widths is too large for the screen, data is displayed in a linear format.

---

* Prime INFORMATION and uniVerse systems have two synonymous keywords, VERT and VERTICALLY, that override the default columnar format of ACCESS reports. VERT allows you to force a report to be displayed in a vertical format, even if the sum of the column widths is less than the screen width.

You can calculate the width of any report as follows:

- Add the width of all columns to be displayed. Column width is either the width defined in line 10 of the Attribute Definition item, or the length of whatever is used as the column heading (i.e., either the item ID of the Attribute Definition item or, if present, the column heading defined in line 3), whichever is greater.
- Add one blank separator between each column in the report.

Displaying Selected Multivalues

You can further limit the data to be output in a report by specifying that only certain values from multivalued attributes be displayed or printed. Such specifications are called *print limiters*. For example, the attribute TITLE in the ORDERS file is multivalued, since many orders are for more than one title. A print limiter can be used to print only the titles you specify.

A print limiter expression consists of a relational operator and a constant. If no operator is included, = (equal to) is assumed. Print limiter expressions are appended to one of the attributes in the output specification.

The following SORT statement includes the print limiter expression TITLE = "OPERATING SYSTEM CONCEPTS":

```
>SORT ORDERS BY DATE DATE CUST.ID TITLE =
   "OPERATING SYSTEM CONCEPTS"
```

Example 3-7 shows the resulting report.

```
PAGE   1                                          11:42:02  01 NOV 1989

ORDERS Date of Order Customer ID  Title.............................

 10101 09/06/88      AJOHN5396
 10102 09/06/88      BLEAR6803    OPERATING SYSTEM CONCEPTS
 10103 09/06/88      AJOHN5396
 10104 09/06/88      MASHX5777
                                  OPERATING SYSTEM CONCEPTS
 10105 09/07/88      DEDGE6635
 10106 09/07/88      AORLA6098    OPERATING SYSTEM CONCEPTS
 10107 09/07/88      HJOHN7265
                                  OPERATING SYSTEM CONCEPTS
 10108 09/08/88      BLEAR6803    OPERATING SYSTEM CONCEPTS
 10109 09/08/88      MASHX5777
 10110 09/09/88      JBOHA5422    OPERATING SYSTEM CONCEPTS
 10111 09/09/88      AJOHN5396    OPERATING SYSTEM CONCEPTS
 10112 09/12/88      BLEAR6803
                                  OPERATING SYSTEM CONCEPTS
 10113 09/13/88      HJENK7129    OPERATING SYSTEM CONCEPTS
 10114 09/13/88      BLEAR6803
 10115 09/13/88      DEDGE6635    OPERATING SYSTEM CONCEPTS
 10116 09/14/88      JBUCK6488
 10117 09/14/88      JSWEN5398
```

Example 3-7.

Print limiters let you specify which multivalues are to be printed.

The syntax of print limiters is similar to that for selection expressions. The difference is that 1) selection expressions select items, and print limiters specify which values from multivalued attributes are to be output; and 2) selection expressions generally begin with the WITH connective.

Print limiters can also be used with Controlling and Dependent attributes to display only data from Dependent attributes that corresponds to the selected Controlling attributes.

The following query lists all orders, but displays the code and number for only those orders with a code of "N01".

```
>LIST ORDERS BOOKCODE = "N01" QTY
```

Example 3-8 shows the resulting report. BOOKCODE is the Controlling attribute and QTY is the Dependent attribute.

```
PAGE   1                                          12:08:39  01 NOV 1989

ORDERS Book Code Qty
                 *
 10101 N01         2
 10107 N01         9
 10113
 10116
 10122
 10110
       N01        12
 10104 N01         3
 10119
 10102
       N01         1
 10105 N01         1
 10108
 10111
 10114 N01         5
 10117
 10120
 10126
 10103 N01         2
 10106
```

Example 3-8.
Print limiting to a Controlling attribute also limits printing of data in a Dependent attribute.

Copying Selected Items with LIST-ITEM

Two ACCESS verbs, LIST-ITEM and SORT-ITEM, deserve a mention in this chapter. These two verbs copy all the data in specified or selected items. Data is listed just as it is stored in the file, with each attribute displayed on its own line. Multivalues are listed on one line, with each value separated by a value mark and each subvalue by a subvalue mark. LIST-ITEM and SORT-ITEM, in fact, combine the functions of the COPY processor with the ability of ACCESS to select specified items.

Example 3-9 lists the item BLEAR6803 from the CUSTOMERS file.

```
>LIST-ITEM CUSTOMERS 'BLEAR6803'
```

```
PAGE   1                                          10:59:15  01 NOV 1989

    BLEAR6803
001 BILL
002 LEARY
003 34 TREMONT STREET
004 BOSTON
005 MA
006 6175278890
007 74332
008 918-27-3645

>
```

Example 3-9.
LIST-ITEM lists data in items as it is stored in the file.

The SORT-ITEM verb works the same way, except it also lists the items in sorted order by the specified attribute. For example:

```
>SORT-ITEM CUSTOMERS BY STATE BY LAST-NAME (P)
```

The preceding query lists all items in the CUSTOMERS file sorted by state, then by last name, and sends the report to the printer.

The USING Connective

To generate a report from the CUSTOMERS file, the system accesses both the CUSTOMERS dictionary and the CUSTOMERS data file. The USING connective makes it possible to generate a report using the Attribute Definition items in *any* file. This file need not contain a D-pointer (a File Definition item) to the data file.

Let's assume that the sample application includes a data file named CUSTOMERS.INACTIVE that contains information about customers whose accounts are inactive. Instead of creating a second file dictionary for the CUSTOMERS.INACTIVE file, you can use the dictionary of the

CUSTOMERS file to access data in CUSTOMERS.INACTIVE since the structure of both files is the same.

The following query generates a report from the CUSTOMERS.INACTIVE file, using the structure defined by the CUSTOMERS dictionary:

```
>LIST CUSTOMERS.INACTIVE USING DICT CUSTOMERS
  FULL-NAME CITY STATE
```

Example 3-10 shows the resulting report.

```
PAGE   1                                          10:53:20  01 NOV 1989

CUSTOMERS.INACTIVE Full Name..... City........ State

AMCSI5349          MCSIMPLE, AMY      VENICE          CA
CWILL5386          WILLIAMS, CHARLES  ARLINGTON       MA
JCROW5329          CROWLEY, JULIA     NORTHWOOD       NH
CGILL5284          GILLETTE, CRAWFORD MYRTLE BEACH    SC
MRUST5317          RUST, MURRAY       OGUNQUIT        MA
PLOMB5244          LOMBARD, PETER     NEW YORK        NY

6 ITEMS LISTED.

>
```

Example 3-10.
The USING connective uses a different file as the dictionary.

The USING connective applies only to situations where files have an identical or nearly identical structure. The referenced attributes in the CUSTOMERS.INACTIVE file are identical to the same attributes in the CUSTOMERS file. The data stored in the two files, however, is different.

The WITHIN Connective

A file comprises items containing data stored as attributes, values, or subvalues. In addition to these, you can create *subitems* in a file that provide a further description of an existing item. Subitems are useful when you want to create a hierarchical relationship among items in a file.

For example, a file might contain an inventory of parts, some of which you want to break down into their components. The item *servos*, for instance, might comprise subitems for *d.c. motor*, *servo board*, and *servo housing*. The item for *d.c. motor* might break down further into *d.c. motor platform* and *d.c. motor power unit*, and so on. Using the WITHIN connective, a bill of materials can be generated that shows the primary item and also any items that are dependent on it.

Subitems are stored in exactly the same way as are regular items. To make a file item a subitem, its item ID is stored as a value in another attribute in the file. The attribute that contains the item IDs of subitems can be multivalued. Each multivalue in this attribute is an item ID for another item stored in the same file.

In addition, the File Definition item in the dictionary (i.e., the D-pointer to the data file) must contain a vertical correlative code in line 8. The syntax of the vertical correlative code is:

V ; ; *attribute*

where *attribute* is the number of the attribute that contains the item IDs of the subitems.

Use the WITHIN connective to retrieve and list subitems of a specified item. The syntax of the WITHIN connective is:

WITHIN *filename item-ID*

Only one item ID can be specified in the query.

When the WITHIN connective is used, each subitem found is assigned a level number. The item ID specified in the query is level 1. If this item ID has subitems, they are assigned level 2. If level 2's items have subitems, they are assigned level 3, and so on up to a maximum of 20 levels. When output is listed, a column called LEVEL will be included that shows the corresponding levels of the subitems.

In Example 3-11, the multivalued attribute SUB-PROD contains item IDs of the PRODUCT file. The following statement lists the item "A2000-1234" and three levels of subitems referenced by it:

```
>LIST WITHIN PRODUCT 'A2000-1234' PROD# DESC VALUE
  LOCATION SUB-PROD QOH ID-SUPP
```

```
PAGE   1                                          15:03:25  01 NOV  1989

LEVEL Prod #.... Description..... Value. Location.. Sub-Prod.. On Hand

 1    A2000-1234 SERVOS                0.73 R-123-8888 A2001-7811      73
                                                       A2001-8900
                                                       A2001-9112
 2    A2001-7811 D.C. MOTOR            0.55 R-17-1001  A2002-1000     643
                                                       A2002-1023
 3    A2002-1000 D.C. MOTOR PLATFORM   0.73 R-123-8888                 58
 3    A2002-1023 D.C. MTR POWER UNIT   0.73 R-123-1002                 89
 2    A2001-8900 SERVO BOARD           0.12 L-44-1001                 329
 2    A2001-9112 SERVO HOUSING         1.09 L-17-189   A2002-1032     107
                                                       A2002-1566
 3    A2002-1032 HOUSING SEALS         1.02 L-09-1889                 768
 3    A2002-1566 HOUSING PLATES        1.03 L-1-3309   A2004-1111     355
 4    A2004-1111 HOUSING PACKAGE      12.00 R-12-1212                 455

9 ITEMS LISTED.

>
```

Example 3-11.
The WITHIN connective lets you list all subitems of an item.

The item specified by the WITHIN connective is listed as Level 1. Level 2 lists subitems specified in the Level 1 item's SUB-PROD attribute. Level 3 lists any subitems specified in the Level 2 item's SUB-PROD attribute. And one of the Level 3 items even contains a Level 4 subitem.

This chapter described how to specify and select items for inclusion in a report, and how to specify which data is output in the report. The next chapter describes how to format reports.

CHAPTER 4

Formatting Reports

This chapter describes the modifiers that can be used to *format*, or specify the appearance of, an ACCESS report. To format a report, you can specify:

- Headings and footings to appear on each page.
- Totals for numeric data in a specified attribute.
- Breaks in the report each time the data in a specified attribute changes. You can also calculate subtotals for each of these sections.
- Suppression of default column headings, page headings, detail lines, and the end-of-list message.
- Double spacing of items.

A number of ACCESS verbs create specialized report formats. These verbs are described in Chapters 6 and 7.

Table 4-1 summarizes the modifiers (and their equivalent parenthetical options) available for formatting ACCESS reports.

Table 4-1. ACCESS Modifiers.

Modifier	Description
BREAK-ON	Specifies control breaks in a report.
COL-HDR-SUPP	Suppresses the time and date heading, column headings, and end-of-list message. (C) option.

DBL-SPC	Double-spaces items in a report.
DET-SUPP	Suppresses detail output when used with TOTAL or BREAK-ON modifiers. (D) option.
FOOTING	Defines a footing for each page of a report.
GRAND-TOTAL	Prints user-specified text on the grand total line.
HDR-SUPP	Suppresses the default heading. (H) option.
HEADING	Overrides the default heading.
ID-SUPP	Suppresses the display of item IDs.
SUPP	Suppresses the default heading. (H) option.
TOTAL	Calculates and displays totals for the specified attribute.

Some keywords function differently when used in combination with certain ACCESS verbs. These combinations are summarized at the end of this chapter.

The rest of this chapter describes how to use most of the modifiers in the preceding table.

A Sample Report

Example 4-1 shows the first page of output from the following query:

```
>SORT ORDERS BREAK-ON DATE SHORT.TITLE TOTAL
 TOTAL.AMT HEADING " 'F' LIST 'CL'PAGE 'P' " FOOTING
 "COMPANY CONFIDENTIAL" GRAND-TOTAL "FINAL
 TOTAL"
```

The report does not suppress item IDs or detail output, and uses single spacing for multivalued items.

Creating Headings and Footings

Reports created with ACCESS generally include *headings*. By default, a heading consisting of the page number and the current system time and date is printed at the top of every page. The end-of-list message that appears at the

end of a report is also considered part of the heading. ACCESS reports do not by default produce *footings* (text at the bottom of every page).

```
                                ORDERS LIST
PAGE    1
ORDERS Date of Order Short Title Amount....

 10101 09/06/88      DATABASE        $19.90
 10102 09/06/88      OPERATING       $66.20
                     DATABASE
 10103 09/06/88      DATABASE        $19.90
 10104 09/06/88      DATABASE       $929.42
                     OPERATING
                     WRITING
                     WORD

       ***                         $1035.42

 10105 09/07/88      DATABASE        $268.81
                     WRITING
                     WORD
 10106 09/07/88      OPERATING        $56.25
 10107 09/07/88      DATABASE        $727.05
                     OPERATING

       ***                         $1052.11

COMPANY CONFIDENTIAL
```

Example 4-1.
The HEADING and FOOTING modifiers customize the report.

You can specify your own report headings and footings by including the HEADING and FOOTING modifiers in an ACCESS query. Enclose the desired heading or footing text in double quotes and enter it immediately after the appropriate modifier.

For example, the following phrase specifies a heading that reads "JANUARY SALES" and a footing that reads "PRELIMINARY":

```
HEADING "JANUARY SALES" FOOTING "PRELIMINARY"
```

The syntax of the HEADING and FOOTING modifiers is as follows:

HEADING | **FOOTING** " [*text*] [*'options'*] [*text*] [*'options'*] ..."

There are a number of additional parameters that can be included in headings and footings. These are specified with the one- and two-character options shown in Table 4-2.

Table 4-2. HEADING and FOOTING Options.

Option	Description
B	When used with the B option of the BREAK-ON modifier, inserts the current breakpoint value at this position in the report heading or footing.
B*n*	When used with the B option of the BREAK-ON modifier, inserts the current breakpoint value at this position in the report heading or footing, left-justified, in a field of *n* blanks.
C	Centers the heading or footing. When used with the L option, centers the specified line.
D	Inserts the current date in the heading or footing, using the form *dd mmm yyyy*.
F	Inserts the name of the file being accessed.
F*n*	Left-justifies the name of the file being accessed in a field of *n* blanks.
L	Starts a new line in the heading or footing. Use this option to create multiple-line text.
P	Inserts current page number, right-justified in a field of four blanks.
PN	Inserts current page number, left-justified.
P*n*	Inserts current page number, left-justified in a field of *n* blanks.
T	Inserts the current time and date in the heading or footing in the form *hh:mm:ss dd mmm yyyy*. The time is shown in 24-hour format.
''	Prints a single quote in the text of the heading or footing. These are two single quotes.

Include options within the heading or footing text (i.e., within the double quotes) and enclose them singly or as a group in single quotes.

Defining a Heading

The report shown in Example 4-2 prints a heading of three lines.

```
                          CUSTOMER LIST
11:01:17  01 NOV 1989
 PAGE    1
CUSTOMERS... Last Name. Street..... City..... State

MASHX5777    ASH         912A E. OAK STREET    INDIANAPOLIS    IN
JBOHA5422    BOHANNON    126 TREMONT STREET    BOSTON          MA
JBROW6749    BROWN       129 BOYLSTON STREET   BOSTON          MA
JBUCK6488    BUCKLER     26 STONE AVENUE       LINCOLN         IN
DEDGE6635    EDGECOMB    338 BROADWAY          MIAMI           FL
AEDWA5224    EDWARDS     51 BLAIR AVENUE       SUDBURY         MA
HHIGG6849    HIGGINS     54 25TH STREET        OMAHA           NB
HJENK7129    JENKINS     1222 MAIN STREET      INDIANAPOLIS    IN
AJOHN5396    JOHNSON     760 JEFFERSON STREET  LOUISVILLE      KY
HJOHN7265    JOHNSON     45 50TH STREET        OMAHA           NB
BLAMP6196    LAMPSON     344 TREMAIN ROAD      BOSTON          MA
BLEAR6803    LEARY       34 TREMONT STREET     BOSTON          MA
JMASO6378    MASON       226 ROCK ROAD         LINCOLN         IN
AMEAD5619    MEADE       251 BLOWNEY AVENUE    SUDBURY         MA
AORLA5993    ORLANDO     55 VENTURA HIGHWAY    VENICE          CA
JPEER5993    PEERCE      89 RIALTO WAY         LOS ALTOS       CA
RPIER5539    PIERCE      123 W RIDGEWOOD AVE   RIDGEWOOD       NJ
SPIRS5289    PIRS        112 APPLEBEE ROAD     WINSTON         NC
```

Example 4-2.

The HEADING modifier reformats the heading of each page of a report.

The report is generated by the following query:

```
>SORT CUSTOMERS BY LAST-NAME LAST-NAME STREET
  CITY STATE HEADING "CUSTOMER LIST 'CLTL'
  PAGE 'P' "
```

The first line is centered and reads "CUSTOMER LIST", the second line contains the current time and date, and the third line contains the page number of the report, preceded by the word "PAGE". Note that when you define a heading, the usual end-of-list message is automatically suppressed.

The L option causes a new line to begin in the heading after:

1. The first (centered) line.
2. The time and date.
3. The page number.

Defining a Footing

Example 4-3 shows a page from a report with the noncentered heading "CURRENT ORDERS FROM THE ORDERS FILE" and a footing that reads "COMPANY CONFIDENTIAL".

This report was produced by the following query:

```
>SORT ORDERS BY DATE DATE TITLE TOTAL.AMT HEADING
  "CURRENT ORDERS FROM THE 'F' FILE" FOOTING
  "COMPANY CONFIDENTIAL"
```

The F option inserts the name of the file "ORDERS" into the heading.

```
CURRENT ORDERS FROM THE ORDERS FILE
ORDERS Date of Order Title.............................. Amount....

 10101 09/06/88      DATABASE MANAGEMENT SYSTEMS              $19.90
 10102 09/06/88      OPERATING SYSTEM CONCEPTS                $66.20
                     DATABASE MANAGEMENT SYSTEMS
 10103 09/06/88      DATABASE MANAGEMENT SYSTEMS              $19.90
 10104 09/06/88      DATABASE MANAGEMENT SYSTEMS             $929.42
                     OPERATING SYSTEM CONCEPTS
                     WRITING COMMERCIAL APPLICATIONS
                     WORD PROCESSING
 10105 09/07/88      DATABASE MANAGEMENT SYSTEMS             $268.81
                     WRITING COMMERCIAL APPLICATIONS
                     WORD PROCESSING
 10106 09/07/88      OPERATING SYSTEM CONCEPTS                $56.25
 10107 09/07/88      DATABASE MANAGEMENT SYSTEMS             $727.05
                     OPERATING SYSTEM CONCEPTS
 10108 09/08/88      OPERATING SYSTEM CONCEPTS               $236.50
                     WRITING COMMERCIAL APPLICATIONS
 10109 09/08/88      WORD PROCESSING                         $160.86
 10110 09/09/88      OPERATING SYSTEM CONCEPTS              $1301.13
                     WRITING COMMERCIAL APPLICATIONS
COMPANY CONFIDENTIAL
```

Example 4-3.
Both the HEADING and the FOOTING modifiers are used.

Suppressing the Page and Column Headings

The following query produces a report from the CUSTOMERS file and suppresses the default column headings and the default heading at the top of each page:

```
>LIST CUSTOMERS LAST-NAME FIRST-NAME STATE
  COL-HDR-SUPP
```

When the page and column headings are suppressed and no other formatting (such as a footing) is defined, the report displays data only. The report is shown in Example 4-4.

```
HJENK7129    JENKINS     HAROLD        IN
JBOHA5422    BOHANNON    JOHN          MA
JBROW6749    BROWN       JAMES         MA
JBUCK6488    BUCKLER     JULIE         IN
BLEAR6803    LEARY       BILL          MA
JMASO6378    MASON       JULIA         IN
AORLA5993    ORLANDO     AMY           CA
SPIRS5289    PIRS        SANDRA        NC
MASHX5777    ASH         MARY          IN
AEDWA5224    EDWARDS     ANTHONY       MA
JPEER5993    PEERCE      JAN           CA
RPIER5539    PIERCE      RICK          NJ
AJOHN5396    JOHNSON     ANNE          KY
HJOHN7265    JOHNSON     HENRY         NB
HHIGG6849    HIGGINS     HENRY         NB
DEDGE6635    EDGECOMB    DAVID         FL
BLAMP6196    LAMPSON     BOB           MA
AMEAD5619    MEADE       ANDREW        MA

>
```

Example 4-4.
The COL-HDR-SUPP modifier suppresses the default column headings as well as the default page heading.

Calculating Totals

Totals for the data in a specified attribute can be generated at the end of a report by including the TOTAL modifier in the query. The syntax for the TOTAL modifier is:

TOTAL *attribute* [*limiters*]

attribute is the name of the attribute.

limiters totals only the data that matches the specified criteria. The criteria are expressed by a relational operator and a literal value enclosed in double quotes or backslashes.

```
PAGE   3                                          11:02:44  01 NOV 1989

ORDERS Customer ID  Title............................. Amount....

 10101 AJOHN5396    DATABASE MANAGEMENT SYSTEMS              $19.90

 10107 HJOHN7265    DATABASE MANAGEMENT SYSTEMS             $727.05
                    OPERATING SYSTEM CONCEPTS

 10113 HJENK7129    OPERATING SYSTEM CONCEPTS               $373.74
                    WORD PROCESSING

 10116 JBUCK6488    WORD PROCESSING                          $45.96

 10122 JBUCK6488    WRITING COMMERCIAL APPLICATIONS          $49.00

 10110 JBOHA5422    OPERATING SYSTEM CONCEPTS               $1301.13
                    WRITING COMMERCIAL APPLICATIONS
                    WORD PROCESSING
                    DATABASE MANAGEMENT SYSTEMS


   ***                                                     $5233.42

23 ITEMS LISTED
```

Example 4-5.
The TOTAL modifier totals data in the specified attribute.

At the end of the report, the total appears in the column of the specified attribute. The total is preceded and followed by a blank line and is identified by three asterisks in the item ID column.

For example, to produce a weekly sales report and total the sales you might enter:

```
>LIST ORDERS CUST.ID TITLE TOTAL TOTAL.AMT DBL-SPC
```

Example 4-5 shows the last page of the report.

The following query totals only sales larger than $250.00:

```
>LIST ORDERS CUST.ID TITLE TOTAL TOTAL.AMT > "250"
 DBL-SPC
```

Example 4-6 shows the last page of the report.

```
PAGE   3                                        10:41:14  01 NOV 1989

ORDERS Customer ID  Title.............................. Amount....

 10101 AJOHN5396    DATABASE MANAGEMENT SYSTEMS

 10107 HJOHN7265    DATABASE MANAGEMENT SYSTEMS              $727.05
                    OPERATING SYSTEM CONCEPTS

 10113 HJENK7129    OPERATING SYSTEM CONCEPTS                $373.74
                    WORD PROCESSING

 10116 JBUCK6488    WORD PROCESSING

 10122 JBUCK6488    WRITING COMMERCIAL APPLICATIONS

 10110 JBOHA5422    OPERATING SYSTEM CONCEPTS               $1301.13
                    WRITING COMMERCIAL APPLICATIONS
                    WORD PROCESSING
                    DATABASE MANAGEMENT SYSTEMS


   ***                                                     $3600.15

23 ITEMS LISTED.
```

Example 4-6.
Totalling can be limited to just the data you specify.

$3600.15 is the total figure for all sales larger than $250.00.

Note that data in the TOTAL.AMT column prints only for items that match the selection criteria.

Formatting the Total Line

The preceding section demonstrated the default appearance of a grand total line in a report. The GRAND-TOTAL modifier can be used to specify a different format.

The syntax of the GRAND-TOTAL modifier is:

GRAND-TOTAL " [*text*] [' *options* '] [*text*] "

text is the string to appear in place of the three asterisks in the item ID column of the total line. Enclose this parameter in double quotes. Text can appear on either side of the specified options.

options are some combination of the following, enclosed in single quotes:

U Underlines the line above the totalled fields with equal signs (=).

L Suppresses the blank line that precedes the total line. This option is ignored if you use the U option.

P Begins a new page for the total line.

options must be specified within single quotes and must be entered somewhere within the double quotes, whether text is included or not.

Let's assume you want to reformat the total line shown in example 4-6. The following query identifies the total line with the text "LARGE ORDER TOTAL:", includes underlining above the total figure, and prints it on a separate page:

```
>LIST ORDERS CUST.ID TITLE TOTAL TOTAL.AMT > "250"
  GRAND-TOTAL "LARGE ORDER TOTAL: 'UP' " DBL-SPC
```

Example 4-7 shows the last page of the report.

```
PAGE   4                                          11:43:26  01 NOV 1989

ORDERS Customer ID  Title.............................. Amount....

                                                          ==========
LARGE ORDER TOTAL:                                          $3600.15


23 ITEMS LISTED.

>
```

Example 4-7.
A customized grand total line.

Breaking on Attribute Values

The BREAK-ON modifier divides an ACCESS report into sections, according to the data in the specified attributes. This modifier causes ACCESS to create a new section (indicated by three asterisks, or by any other specified text) whenever the data changes. For example, in a report that lists company personnel by department, each department might appear in a separate section.

The point at which a new section begins is called a *control break*. Up to 15 different control breaks can be specified in a single ACCESS query. The highest level break is the first one specified.

Totals can also be included at the end of each section in the report. This feature is described later in this section.

The syntax for the BREAK-ON modifier is:

BREAK-ON *attribute* [" [*text*] ['*options* '] [*text*] "]

attribute is the attribute on whose changing data a new section begins. ACCESS determines the change by testing the first 24 characters of the data, from left to right.

The specified attribute is automatically included as one of the columns in the report; it need not be entered separately as an output specification.

text is printed in the attribute column when a control break occurs. This text replaces the three asterisks that are output by default. Enclose the specified text in double quotes.

options affect the processing of the control break. These include:

B Inserts data from the attribute specified in the report heading if the B option was used with the HEADING modifier. Use this option with only one BREAK-ON modifier per query.

D Suppresses the break line data if only one new line was output since the last control break.

L Suppresses the blank line preceding the break line. This option has no effect when specified with the U option.

P Starts a new page after each control break.

R Forces all data associated with a control break to appear on the same page.

U Underlines all total fields for each control break.

V Inserts the current data for the attribute into the text on the control break line.

Enclose options in single quotes, along with any accompanying text. *options* and *text* must be enclosed in double quotes.

For example, let's assume you are generating a report about classes of children in a school. The following query sorts the children's names alphabetically and divides the report into alphabetical sections by the teacher's last name. The listing for each teacher begins on a new page:

```
>SORT STUDENTS BY TEACHER BY NAME NAME BREAK-ON
  TEACHER " 'P' "
```

Example 4-8 shows the first page (section) of the report.

```
PAGE   1                                          11:05:32  01 NOV 1989

STUDENTS.. Name.............. Teacher...

4504       HARMON, GEORGE     JONES
4505       PORTNOY, SYLVIA    JONES
4506       RATNER, HEIDI      JONES
4503       WATKINS, JOHN      JONES
4507       WELCONE, JIM       JONES

                              ***
```

Example 4-8.
Using the BREAK-ON modifier.

Including Totals in a Control Break

The TOTAL modifier (described earlier in this chapter) can be used in combination with the BREAK-ON modifier to total the data for a specified attribute in each section of a report. When the TOTAL modifier and the BREAK-ON modifier are used together, the following occurs:

1. A subtotal appears in the report for the attribute specified with the TOTAL modifier each time the value of the attribute specified with the BREAK-ON modifier changes.
2. A total of all of the subtotals appears at the end of the report.

Example 4-9 shows subtotals for the class dues paid by two classes:

```
>SORT STUDENTS BY TEACHER BY NAME NAME BREAK-ON
  TEACHER TOTAL DUES
```

```
PAGE   1                                          11:05:55  01 NOV 1989

STUDENTS.. Name.............. Teacher... Dues....

4504       HARMON, GEORGE     JONES         $5.00
4505       PORTNOY, SYLVIA    JONES         $2.00
4506       RATNER, HEIDI      JONES         $9.00
4503       WATKINS, JOHN      JONES         $4.00
4507       WELCONE, JIM       JONES         $7.00

                              ***          $27.00

4513       BENNETT, NANCY     KUBOVY        $3.00
4515       BERMAN, ANGELA     KUBOVY        $5.00
4514       CURRAN, DIANE      KUBOVY        $3.00
4516       JOHNSON, MEG       KUBOVY        $7.00
4517       KESTER, MICHAEL    KUBOVY        $9.00
4518       TERNY, KEITH       KUBOVY        $6.00

                              ***          $33.00
```

Example 4-9.
Using the BREAK-ON and TOTAL modifiers together.

Suppressing Detail Lines

When the TOTAL and BREAK-ON modifiers are used together (or when the TOTAL modifier is used on its own), the DET-SUPP modifier can be used to suppress all of the detail lines in the report. In this case, only the subtotal and total lines are displayed.

The following query suppresses the detail lines for the report:

```
>SORT STUDENTS BY TEACHER BREAK-ON TEACHER
  TOTAL DUES DET-SUPP
```

```
PAGE   1                                        11:06:34  01 NOV 1989

STUDENTS.. Teacher... Dues....

           JONES       $27.00

           KUBOVY      $33.00

           MOYERS      $46.00
***                   $106.00

18 ITEMS LISTED.

>
```

Example 4-10.
The DET-SUPP modifier lists only subtotal and total lines; other lines are suppressed.

Special Uses of Formatting Modifiers

When used in combination with certain verbs, some of the formatting modifiers described in this chapter function differently. These differences occur in the following areas:

- The HDR-SUPP, HEADING, ID-SUPP, and SUPP modifiers work differently with the T-DUMP and S-DUMP verbs. HDR-SUPP, SUPP, and HEADING affect the creation of the tape label, and ID-SUPP suppresses the listing of item IDs on the screen as items are copied to tape.
- The ID-SUPP modifier works differently with the REFORMAT and SREFORMAT verbs, determining what the item IDs will be in the new file.

- When used in forms generation statements, the BREAK-ON, TOTAL, and GRAND-TOTAL modifiers have a more specialized role. In addition, the HEADING and FOOTING modifiers function in a more limited way, and the ID-SUPP modifier is superfluous.

These special functions are described fully in Chapters 6 and 7.

CHAPTER 5

Using Select-Lists

The preceding chapters describe how to enter ACCESS queries that define the contents, scope, and format of a report. This chapter describes a group of ACCESS verbs that create and manipulate *select-lists*. Select-lists provide added flexibility in extracting information from a database and in generating reports.

A select-list is a list of data elements created by the ACCESS query. These elements identify file items whose data meets specified criteria. Elements in the select-list can then be used as input to another process or query, or the select-list can be saved for use later. For example, you might create a select-list from the CUSTOMERS file that selects the item-IDs as input to a PICK/BASIC program. Or you could save the select-lists in the POINTER-FILE, edit it with the Editor, then reactivate it and use it in another query.

Why use a select-list when you can just as well use a selection expression to select items? Because when you are retrieving data, especially from large files, it can take some time to process a selection expression—much more time than it takes to retrieve data using a list of specified item IDs. Also, select-lists can be saved and reused as many times as you like; you don't have to keep entering the same selection expression again and again to retrieve the same subset of data.

Creating a select-list can be compared to using a selection expression in a query that generates a report. In both cases, only those items meeting the specified criteria are selected. Using a selection expression to generate a

report, however, is a one-step operation. A select-list, on the other hand, merely *identifies* the specified items. How the list will then be used (the second step) is up to the user.

Select-lists can be used to reference data in any file, not just the file from which the select-list was generated. For example, you might create a select-list from the ORDERS file of all customers whose accounts are overdue by more than $100. You could then use the select-list to access the CUSTOMERS file to extract these customers' addresses for a mailing.

A select-list is always temporary: that is, it is available only for the execution of the next command, query, or program. This next operation must immediately follow the query that created the select-list; if it does not, the select-list is lost. For example, the following query creates a select-list from the CUSTOMERS file dictionary; the items in this list are then processed by the Editor:

```
>SELECT DICT CUSTOMERS WITH A/AMC = "2"


4 ITEMS SELECTED.
>ED DICT CUSTOMERS
LAST-NAME
TOP
```

The preceding SELECT statement creates a select-list of all Attribute Definition items in the CUSTOMERS dictionary that have an attribute number of 2. The ED verb then invokes the Editor on these items starting with the first item, LAST-NAME. After the first item is filed (or exited), the next item in the select-list is automatically displayed, and so on. When the user has finished editing the items, the select-list is automatically cancelled.

You can make a select-list permanent by explicitly saving it to the POINTER-FILE with the SAVE-LIST verb. The POINTER-FILE is a special file in the SYSPROG account that is used to store saved select-lists. Every account on the system has a Q-pointer to this file, so once a select-list has been saved, it is accessible to other users and can be retrieved and used at any time. The GET-LIST, EDIT-LIST, COPY-LIST, and DELETE-LIST verbs are used to access and manipulate saved select-lists.

It is possible to create a local pointer-file in any account. Saved lists stored in a local pointer-file are available only to users who have access to files in that account.

To create a local pointer-file in any account, create a file dictionary and change the D/CODE of the D-pointer in the Master Dictionary from "D" to "DC". This creates a file that can point to compiled code (such as PICK/BASIC object code) as well as to select-lists.

Processors That Use Select-Lists

This section summarizes the way that various system processes interact with select-lists.

PICK/BASIC

Select-lists are available to PICK/BASIC programs via the READNEXT statement. Such a select-list overrides the first SELECT statement in a PICK/BASIC program. The select operation can also be initiated by a PICK/BASIC SELECT statement or an EXECUTE statement in combination with one of the ACCESS select-list verbs.

ACCESS

An active select-list does not automatically take precedence over parameters specified in the next query. A select-list is ignored, for instance, if the next query includes an explicit item-list or an item selection expression. On the other hand, you *can* use a selection expression (using WITH) in the subsequent query to access a subset of the items contained in the active select-list.

An explicit item list or an item selection expression always takes precedence over an active select-list. For example,

```
>SELECT CUSTOMERS WITH LAST-NAME > "NARDONE"

4 ITEMS SELECTED.
>LIST-ITEM CUSTOMERS 'TABBO5729' 'GLEBO5687'
```

Since the LIST-ITEM query includes two explicit item IDs, the select-list is ignored and a report is generated only for items TABBO5729 and GLEBO5687.

Runoff

The items in a select-list can be inserted into Runoff text via the READNEXT command.

TCL-II Verbs

TCL-II verbs interact with select-lists just as ACCESS verbs do. For example, you might use a select-list with the COPY verb to write to tape a subset of items in a file.

Creating Select-Lists

A number of ACCESS verbs can be used to create select-lists. The two principal verbs are SELECT and SSELECT. SELECT creates a select-list of the items that match the specified criteria; the elements in the select-list are arranged in the order in which the items are currently stored in the file. SSELECT works the same way except that it sorts the elements by the specified attribute. SELECT is thus analogous to LIST, and SSELECT to SORT.

If an output specification is specified, data elements from the output attributes are used to make up the select-list instead of item IDs. Each value in a multivalued attribute is stored as a separate element.

Table 5-1 summarizes the ACCESS verbs that create select-lists.

Table 5-1. Select-List Verbs.

Verb	Description
SELECT	Selects items that meet the specified criteria.
SSELECT	Selects and sorts items that meet the specified criteria.
FORM-LIST	Creates a select-list from data in the specified items rather than from their item IDs.
QSELECT	A synonym for FORM-LIST.
NSELECT	Selects items that are in an active select-list but are not in the specified file. Not implemented on all systems.

SELECT Syntax

The SELECT verb uses standard ACCESS query syntax:

SELECT [**DICT**] *filename* [*item-list*] [*selection*] [*output*]

SSELECT uses the same syntax as SELECT and additionally allows you to define one or more sort expressions. Without any sort expressions, SSELECT sorts the selected items in ascending order by item ID.

Creating a Select-List (SELECT)

Let's assume you have an ACCOUNTS file that includes (among others) attributes for CUST.ID, ORDER.NO, DATE, BALANCE.DUE, AMT.PAID, and STATUS. You want to select from the ACCOUNTS file only those items whose balances are still unpaid. You then want to generate a report listing all unpaid accounts. This is a two-step process: first you select the items you want with SELECT, then you generate the report with another ACCESS verb.

First you want the program to select only those items whose balances are still unpaid. The STATUS attribute contains "PD" if payment has been received; if the account has not been paid, STATUS contains no value. Create a SELECT statement that selects only those items which contain no value in the attribute STATUS:

```
>SELECT ACCOUNTS WITH STATUS = " "

58 ITEMS SELECTED.
>
```

The set of quotation marks specifies a null value in the STATUS attribute. A message indicates that 58 items were selected, and the TCL prompt reappears. You must now use the select-list in the next command, or you will lose it! You might enter the following to print out a sorted report:

```
>SORT ACCOUNTS BY CUST.ID CUST.ID DATE
  BALANCE.DUE (P)
```

Only the items identified by the select-list will be sorted and listed. The select-list is no longer available and cannot be reused.

If, however, you had saved the select-list immediately after you had created it, it would be available whenever you wanted to use it. For details about how to save a select-list, see the section, "Saving a Select-List," later in this chapter.

Another way to use select-lists is in BASIC programs. You might run a PICK/BASIC program that uses the select-list in the preceding example to generate invoices for all unpaid items. Once the items have been selected, the program can execute a forms generation statement to print the invoices. Since the select-list of unpaid accounts is active, no further selection processing is necessary. (Forms generation verbs are discussed in Chapter 7.)

Listing Items Not Included in a Select-List (NSELECT)

Some systems support the NSELECT verb. NSELECT is used to determine which elements in an active select-list are *not* contained in another file. Creating a select-list with NSELECT is a two-step process: first you use one of the regular list-creating verbs to select elements from file A, then you use NSELECT to select only those elements of the active select-list that are *not* contained in file B. The second select-list (the one created with NSELECT) is now the only active one and can be used, for example, to list items in file A that are not in file B.

NSELECT uses the following syntax:

NSELECT *filename*

The following example first creates a select-list of all of the items in the ORDERS file, then creates a second select-list from the items on this list that are not in the ORDERS.PAID file. The result is a list of those items in the ORDERS file that are currently unpaid:

```
>SELECT ORDERS

347 ITEMS SELECTED.
>NSELECT ORDERS.PAID

256 ITEMS SELECTED.
>
```

The unpaid order items can then be listed with the following query:

```
>LIST ORDERS
```

NSELECT can be used only when another select-list is currently active. Thus NSELECT must be used immediately after a select-list has been created with one of the list-creating verbs or has been made active with GET-LIST.

Creating a Select-List from Data (FORM-LIST)

FORM-LIST (and its synonym QSELECT) creates a select-list containing all or selected data in the specified file rather than from the item IDs. FORM-LIST uses the following syntax:

FORM-LIST [**DICT**] *filename items* [**(***attr#* **)**]

You must specify either an explicit item-list or all items in the file (indicated by an asterisk). If you specify an item, the select-list will comprise all data from all attributes in that item. Each line of data—that is, each attribute—of the item specified becomes a separate element in the select-list. If you specify several items in the FORM-LIST statement, the select-list will comprise all of the data from all of the items specified, listed in the order of *items*.

Each value in a multivalued attribute is stored as a separate element in the select-list. Example 5-1 shows some items from the ORDERS file that contain multivalues in Attributes 3 and 4.

```
PAGE   1                                          13:22:39  01 NOV 1989

    10102
001 BLEAR6803
002 7555
003 N02]N01
004 3]1

    10104
001 MASHX5777
002 7555
003 N01]N02]QR01]QR02
004 3]5]1]34

    10105
001 DEDGE6635
002 7556
003 N01]QR01]QR02
004 1]4]7
```

Example 5-1.
Items in the ORDERS file have multivalues in Attributes 3 and 4.

Example 5-2 shows a select-list of all the data in Attribute 3, BOOKCODE. Each multivalue is stored as a separate element.

This select-list can be created by either of the following two queries:

```
>SELECT ORDERS BOOKCODE
>FORM-LIST ORDERS * (3)
```

```
    BKCODE
001 QR01
002 N02
003 QR01
004 QR02
005 N01
006 N01
007 N02
008 QR01
009 QR02
010 N02
011 N01
012 N01
013 N02
014 N02
015 QR02
016 QR02
017 N02
018 N01
019 N01
020 QR01
```

Example 5-2.
The select-list BKCODE lists each multivalue separately.

You can see from the preceding example that using FORM-LIST you can specify that only data contained in a certain attribute be used in the select-list. *attr#* is the *number* of the desired attribute (you cannot specify an attribute by its *name*).

Example 5-3 shows how you might use a select-list to reference a second file. Let's assume that Attribute 1 in the ORDERS file contains customer IDs that match the item IDs in the customers file. A FORM-LIST statement first creates a select-list of customer IDs from the data contained in Attribute 1. Then a SORT-LABEL statement creates a mailing list from the customers file of the active customers selected by FORM-LIST, to inform them of several new books that are available.

```
>FORM-LIST  ORDERS  *  (1)


226 ITEMS SELECTED.
>SORT-LABEL  CUSTOMERS  BY  LAST-NAME  FULL-NAME  STREET  CITY-ST
    ZIP  ID-SUPP

?3,4,2,6,20,3,C
?NAME
?ST
?CITY
?ZIP
     .
     .
     .
```

Example 5-3.

A list of item IDs is formed from the customer IDs stored in Attribute 1 of the ORDERS file. The list is then used to reference the CUSTOMERS file.

You can also form a select-list from data in an attribute by specifying the name of the attribute in the SELECT query. Thus, the select-list created by the preceding FORM-LIST command could also be created by the following SELECT query:

```
>SELECT ORDERS CUST.ID
```

Saving a Select-List

The SAVE-LIST verb gives a select-list a name and stores it as an item in the POINTER-FILE. This can be either the system POINTER-FILE (in the SYSPROG account) or a local pointer-file. If the system POINTER-FILE is used, the select-list is accessible to all other accounts on the system. A saved

select-list can be removed from the POINTER-FILE with the DELETE-LIST verb.

To save a select-list, enter the SAVE-LIST verb immediately after creating the list. For example:

```
>SELECT ORDERS WITH AMOUNT > "25000"

23 ITEMS SELECTED.
>SAVE-LIST LARGE-ORDERS
LIST 'LARGE-ORDERS' SAVED - 1 FRAMES USED.
```

The system reports the number of frames used to store the LARGE-ORDERS select-list, in this case, one frame.

☞ If a saved select-list of the same name already exists, the new one overwrites it without asking for confirmation.

Working with Saved Select-Lists

There are several verbs that can be used to work with saved select- lists. These verbs are summarized in Table 5-2 and described in the sections that follow.

Table 5-2. Saved Select-List Verbs.

Verb	Description
COPY-LIST	Copies a saved list to the terminal screen, the printer, another saved list, or a file item.
DELETE-LIST	Removes a saved list from the POINTER-FILE.
EDIT-LIST	Invokes the Editor on a saved list.
GET-LIST	Retrieves a saved list and makes it available to the subsequent command, query, or program.

All of these verbs use the following syntax:

verb list-name

list-name is the name assigned to the select-list, when it was saved with the SAVE-LIST verb.

Retrieving a Saved Select-List

GET-LIST retrieves a saved select-list and activates it: that is, it makes it available to a subsequent command, query, or program. You can execute only one such command on a retrieved select-list, however. If you want to execute more than one command, you must retrieve the saved list again.

Let's assume that you previously created and saved a select-list called ORDERS.FEB that identifies all orders taken during the month of February. The following two commands retrieve the saved list from the POINTER-FILE and send a report on February orders to the printer:

```
>GET-LIST ORDERS.FEB

85 ITEMS SELECTED.
>SORT ORDERS BY-DSND TOTAL.AMT REPORT (P)
```

Copying a Saved Select-List

COPY-LIST is similar to the COPY verb, except that it copies saved select-lists instead of file items. COPY-LIST can be used to display a saved list on the terminal screen or send it to the printer, rename a saved list, or copy a saved list to a file item.

COPY-LIST uses the following subset of the options available with the COPY verb:

Table 5-3. COPY-LIST Options.

Option	Description
D	Deletes the original saved list after it is copied to another saved list or to a file item. This option is useful when you want to rename a saved list.
N	Disables paging when copying the saved list to the terminal.
O	Overwrites an existing saved list.
P	Copies the saved list to the printer.
T	Copies the saved list to the terminal screen.
X	Copies the saved list to the printer or terminal screen in hexadecimal format.

After you enter COPY-LIST, the system displays the prompt:

```
TO :
```

Enter the destination as follows:

{ *dest-list* | [([**DICT**] *filename*)] *item-ID* }

You can copy the saved list either to another saved list or to a file item.

For example, let's assume there is a saved list on your system called LARGE-ORDERS. This list was created in 1988. At the beginning of 1989, you might want to rename this list LARGE-ORDERS-88, as follows:

```
>COPY-LIST LARGE-ORDERS (D)
TO :LARGE-ORDERS-88
    1 LARGE-ORDERS TO LARGE-ORDERS-88
```

You could then send this list to the printer with the following command:

```
>COPY-LIST LARGE-ORDERS-88 (P)
```

☞ If the saved list is larger than 32 K, you cannot copy it to a file item. Instead, you get the following error message:

[196] '*list*' IS TOO LARGE TO BE AN ITEM.

Editing a Saved Select-List

EDIT-LIST invokes the Editor on a saved select-list. You can then make edits to refine the list (typically, by deleting or adding lines).

Let's assume that you just received overdue payments from three customers. The following example calls up a saved list named OVERDUE.CUSTOMERS and uses the editor to delete the item IDs on lines 5, 6, and 7:

```
>EDIT-LIST OVERDUE.CUSTOMERS
OVERDUE.CUSTOMERS
TOP
.G5
005 LERC4MAPLL
.DE 3
.FI
[243] LIST 'OVERDUE.CUSTOMERS' SAVED - 1 FRAMES USED
```

CHAPTER 6

Specialized Processing

This chapter describes five groups of ACCESS verbs that process data in specialized ways:

1. Creating labels.

   LIST-LABEL SORT-LABEL

   These verbs display file items in label, or block, format. Users can define the exact placement of these blocks on the screen or page. This format is especially useful for printing mailing labels.

2. Generating statistics on data in files.

   COUNT SUM STAT

   These verbs do not format and display the data in file items. Instead, they return statistics about the data, such as the total for a numeric attribute in selected items. These verbs allow users to perform simple analyses on a database.

3. Generating file statistics.

   FILE-TEST ISTAT CHECK-SUM
   HASH-TEST

   Like the verbs in the previous item, these verbs do not format and display data in file items. They return statistical information about a file or about specified information in a file. This information includes a byte count, a bit count, distribution of items in a file, etc.

4. Copying items to and from tape.

   T-DUMP S-DUMP T-LOAD

   These verbs copy items to and from magnetic tape. They can be used to create and maintain permanent records of a database and to store selected data off-line.

5. Restructuring output.

   REFORMAT SREFORMAT

   These verbs actually alter the structure of file items. They can be used to copy restructured items to magnetic tape, to another file, or within the same file. Restructuring items in this way is useful for rearranging existing files.

Table 6-1 summarizes the ACCESS verbs that do special processing.

Table 6-1. Special Processing Verbs.

Verb	Description
LIST-LABEL	Displays items in label format.
SORT-LABEL	Displays items in label format in sorted order.
COUNT	Counts file items.
SUM	Totals the data in a numeric attribute.
STAT	Displays statistics about a numeric attribute.
CHECK-SUM	Produces check-sum statistics for file items.
ISTAT	Summarizes item distribution in a file.
HASH-TEST	Tests effects of different modulos on item distribution.
T-DUMP	Copies items to magnetic tape.
S-DUMP	Copies items to magnetic tape in sorted order.
T-LOAD	Copies items from magnetic tape to disk.
REFORMAT	Restructures items and directs output to magnetic tape, another file, or within the same file.
SREFORMAT	Restructures items in sorted order. (See REFORMAT.)

Printing Labels

The LIST-LABEL and SORT-LABEL verbs output items in individual blocks or *labels*. Labels can be used for mailings or to identify inventory items. The size of the labels and their position on the page can be set by the user. This makes it possible to output items on preprinted paper labels of any size.

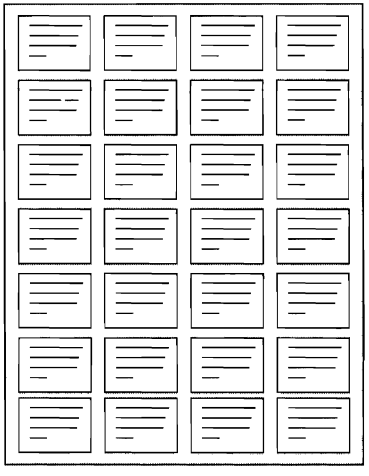


Figure 6-1.
The LIST-LABEL verb outputs items onto preprinted labels.

In contrast, the LIST and SORT verbs rely on the Attribute Definition items in the file dictionary to determine the basic format of a report.

LIST-LABEL and SORT-LABEL Syntax

The syntax for the LIST-LABEL verb is identical to that of the LIST verb. That is, you can include an explicit item list, selection expressions, output specifications, modifiers, and options. Note, however, that the COL-HDR-SUPP and DBL-SPC modifiers work differently for reports in label format. The COL-HDR-SUPP modifier produces a continuous form report (one with no page breaks); it also suppresses the default heading on each page of the report. The DBL-SPC modifier has no effect.

The SORT-LABEL verb has the same syntax as the LIST-LABEL verb, with the addition of sort expressions. The sorting capability is especially useful with label formats because it allows you to produce mailing labels in alphabetical order (for example, by last name) or in numeric order (for example, by zip code).

Defining a Label Format

After the LIST-LABEL verb is entered, the system displays a question mark prompt:

```
>LIST-LABEL CUSTOMERS FULL-NAME STREET CITY-ST
  ZIP

?
```

At this prompt, specify the label format on a single line as follows:

count, rows, skip, indent, size, space [, **C**]

count	is the number of labels (items) across each page or screen.
rows	is the number of lines printed for each label. Remember to count the item ID as one line. The item ID is automatically included in the labels unless you use the ID-SUPP modifier in the query.
skip	is the number of lines to skip vertically between labels.
indent	is the number of indented spaces from the left margin to the first column of labels. 0 is a valid response.
size	is the maximum width for each attribute line (in other words, the width of each label in columns).
space	is the number of horizontal spaces between labels.
C	specifies that null attributes should not be printed. Otherwise, null attributes appear as all blanks. This parameter is optional.

The total width specifications cannot exceed the page width (normally 80 characters for terminals, and 80 or 132 characters for printers). Determine the available width using the following formula:

```
(count * (size + space) + indent) <= (page width)
```

page width is the value defined by the TERM verb for the terminal or printer.

If a value other than zero is specified for the *indent* parameter, the system prompts you to define headers for each row in a label:

```
?
```

The number of ? prompts corresponds to the value entered earlier for *rows*. At each prompt, type the desired header and press the RETURN key. (To avoid defining a header, simply press the RETURN key.) These headers will appear at the left margin in the *indent* area of the listing.

Example 6-1 shows partial output for a roll of mailing labels that has three labels in each horizontal row. The labels were generated by the following SORT-LABEL statement:

```
>SORT-LABEL CUSTOMERS BY LAST-NAME FULL-NAME
   STREET CITY-ST ZIP ID-SUPP COL-HDR-SUPP

?3,4,2,8,18,4
?NAME
?ST
?CITY
?ZIP
```

```
NAME    ASH, MARY              BOHANNON, JOHN         BROWN, JAMES
ST      912A E. OAK STREET     126 TREMONT STREET     129 BOYLSTON ST
CITY    INDIANAPOLIS,IN        BOSTON,MA              BOSTON,MA
ZIP     98754                  21300                  21300


NAME    BUCKLER, JULIE         EDGECOMB, DAVID        EDWARDS, ANTHONY
ST      26 STONE AVENUE        338 BROADWAY           51 BLAIR AVENUE
CITY    LINCOLN,IN             MIAMI,FL               SUDBURY,MA
ZIP     02144                  39007                  01776


NAME    HIGGINS, HENRY         JENKINS, HAROLD        JOHNSON, ANNE
ST      54 25TH STREET         1222 MAIN STREET       760 JEFFERSON ST
CITY    OMAHA,NB               INDIANAPOLIS,IN        LOUISVILLE,KY
ZIP     39977                  48865                  99475


NAME    JOHNSON, HENRY         LAMPSON, BOB           LEARY, BILL
ST      45 5OTH STREET         344 TREMAIN ROAD       34 TREMONT STREET
CITY    OMAHA,NB               BOSTON,MA              BOSTON,MA
ZIP     39977                  74332                  74332

>
```

Example 6-1.
Labels generated by the SORT-LABEL verb.

The preceding statement defines a label format that prints three labels across. Four lines (attributes) are displayed for each label, two vertical lines are skipped between each row of labels. There is an initial indent of eight spaces, a maximum of eighteen characters for each attribute, and four horizontal spaces between each label. The ID-SUPP modifier suppresses the inclusion of item IDs, and the COL-HDR-SUPP modifier suppresses the default page headings.

Before you send labels to the printer, you should use the SP-ASSIGN command to assign the correct forms queue to your account. A form queue can be a particular printer that's loaded with special paper, such as label stock.

Generating Labels with a Proc

You can greatly simplify the process of generating labels by using a proc that contains a LIST-LABEL or SORT-LABEL command. Procs let you execute lengthy ACCESS statements simply by entering one command. They can also supply responses to prompts, so you don't have to enter all the information required by the ? prompt each time you run off a set of labels.

The labels generated by the SORT-LABEL command discussed in the previous section can also be generated by the following proc called LABELS:

```
001 PQ
002 HSORT-LABEL CUSTOMERS BY LAST-NAME
        FULL-NAME STREET CITY-ST ZIP (IC)
003 STON
004 H3,4,2,8,18,4<
005 HNAME<
006 HST<
007 HCITY<
008 HZIP
009 P
```

When the SORT-LABEL statement in line 2 is executed, the proc supplies the label format (line 4) and all necessary responses to the ? prompt (lines 5, 6, 7, and 8). All the user need do is enter the command LABELS to produce the labels shown in Example 6-1. A more generalized version of this proc might prompt the user for the filename and generate labels for any file containing these attributes.

Generating Statistics on Data

The COUNT, SUM, and STAT verbs generate statistics for the data in a file. These three verbs produce a short summary rather than a formatted listing of the data stored in the file.

These verbs all permit the use of selection expressions, but none of the modifiers used to format reports are applicable. Because the COUNT, SUM, and STAT verbs produce only one or two lines of information, there is no need to specify a heading, footing, double spacing, or any other formatting for the report.

The statistics verbs automatically include all data in multivalued attributes in their calculations.

Counting File Items

The COUNT verb displays the number of file items that meet the specified conditions. This makes it possible to count the number of items in a file, the frequency with which a constant occurs in a file, and the number of items that meet a condition expressed with relational operators. This verb can be useful for determining how large a report will be, especially when the file itself is large.

For example, you could count any of the following:

- The number of customers in the CUSTOMERS file:

```
>COUNT CUSTOMERS

543 ITEMS COUNTED
```

- The number of customers who live in Indiana:

```
>COUNT CUSTOMERS WITH STATE "IN"

36 ITEMS COUNTED
```

- The number of orders in the ORDERS file that were placed after December 31, 1987:

```
>COUNT ORDERS WITH DATE > "12/31/87"

256 ITEMS COUNTED.
```

- The number of orders for which at least two copies of one of the titles ordered were requested (QTY is the attribute in the ORDERS file that reflects the number of books sold for each title):

```
>COUNT ORDERS WITH QTY GE "2"

112 ITEMS COUNTED.
```

Totalling a Numeric Attribute

The SUM verb calculates a total for the data in a specified numeric attribute. For example, you can generate a total for dollar amounts such as sales figures or for quantities such as the number of books sold.

Let's assume that a new advertising campaign was initiated on April 13, 1989 and you want to know how many books were sold during the subsequent three months, as compared to the three months prior to the campaign. The following two queries provide this information:

```
>SUM ORDERS QTY WITH DATE GE "04/13/89" AND LE
     "07/13/89"

TOTAL OF Qty = 578

>SUM ORDERS QTY WITH DATE GE "01/13/89" AND LE
     "04/13/89"

TOTAL OF Qty = 332
```

Generating Statistics for a Numeric Attribute

The STAT verb combines the functions of the COUNT and SUM verbs and also averages the value of a numeric attribute. The result is three values:

1. The total of the data in the numeric attribute.
2. The average value of this data.
3. The number of items selected.

This verb is generally used for analysis rather than reporting.

The following query displays statistics about the sale amounts (AMOUNT) for the book whose code is N01:

```
>STAT ORDERS TOTAL.AMT WITH BOOKCODE = "N01"

STATISTICS OF Amount :
TOTAL = $2524.00   AVERAGE = $315.0050   COUNT = 8
```

Generating File Statistics

The FILE-TEST and ISTAT verbs let you generate statistics on how items are stored (item distribution in groups). The FILE-TEST and HASH-TEST verbs also lets you test a file to see what the effect on item distribution would be if you used different test modulos. FILE-TEST supersedes ISTAT and HASH-TEST, two older commands available on most systems. ISTAT gives statistics on the file as it is currently set up, HASH-TEST shows how the items would be reallocated if you used a given modulo for the file. HASH-TEST perform exactly the same function as ISTAT but on a "what if" basis.

These three commands are not typically used by end users. They are useful for system administrators and applications developers when they want information that will help in reallocating files which have become either too large or too small for the disk space originally allocated to them.

For purposes of illustration, the results of these commands are shown for the default Master Dictionary that is created in a new user's account. The examples are from a system that uses a frame size of 2048 bytes. (Systems with smaller frame sizes use another parameter, *separation*, that allows the number of contiguous frames per group to be set for optimum performance—another factor in determining the best file size.)

Analyzing Current File Structure

FILE-TEST has the following syntax:

FILE-TEST [**DICT**] *filename* [*items*] [*selection*] [*modifiers*]
[(*options*)]

After FILE-TEST is entered, you are prompted to enter either a test modulo, or (on some systems) a test modulo and separation. If all you want to do is obtain statistics on the file as it is currently configured, press the RETURN key at the prompts.

ISTAT uses the same syntax as FILE-TEST. After you enter ISTAT, however, there are no prompts: the report is displayed or printed immediately.

ISTAT and FILE-TEST with no test parameters produce a summary of the current item distribution for a file and analyze the structure of groups within the file. This item distribution summary can help you determine whether the current file structure is the best one for the file.

Example 6-2 shows the result of the ISTAT command for the Master Dictionary of an account.

The histogram shows that items are distributed evenly. The lowest number of items in a group is 28, the highest is 49. The average number of items per group is 40. The average item size is 21 bytes, making the average number of bytes per group 858 bytes. None of the groups require additional overflow frames.

```
>ISTAT MD

FILE= MD MODULO= 7                    15:36:03  01 NOV 1989
FRAMES BYTES ITMS
     1   1006  45 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1    595  28 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1    787  38 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1    898  42 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1    643  33 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1    862  45 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     1   1219  49 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
     7

ITEM COUNT=       280, BYTE COUNT=    6010, AVG.BYTES/ITEM=   21.4
AVG.ITEMS/GROUP= 40.0, STD.DEVIATION=  7.4, AVG.BYTES/GROUP= 858.5
```

Example 6-2.
ISTAT displays a histogram and statistics illustrating current item distribution in a file.

Testing Alternate File Structures

HASH-TEST and FILE-TEST with test parameters both produce a listing similar to that produced by ISTAT shown above, but allow you to see what the distribution of records would be, given any modulo. The file's modulo is

not changed by these commands; they simply calculate the statistical information using the test modulo (or modulo and separation).

Example 6-3 shows the result of the HASH-TEST command run on the same Master Dictionary shown in the preceding section. A test modulo of 3 has been used instead of the actual modulo of 7.

```
>HASH-TEST MD

TEST MODULO:3
```

```
FILE= MD MODULO= 3                    15:44:46  01 NOV 1989
FRAMES BYTES ITMS
     1   1740  78 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
     2   2005  98 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
     2   2265 104 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>!
     5

ITEM COUNT=       280, BYTE COUNT=    6010, AVG. BYTES/ITEM=   21.4
AVG.ITEMS/GROUP= 93.3, STD.DEVIATION= 13.6, AVG.BYTES/GROUP= 2003.3
```

Example 6-3.
HASH-TEST produces a report just like the ISTAT report, but uses a hypothetical modulo.

You can see that items are well distributed across 3 groups, with an average of 93 items in each group; however, the number of bytes in each group is too large and exceeds one frame.

On the other hand, with a test modulo of 79:

```
>HASH-TEST MD

TEST MODULO:79
```

we get the result shown in Example 6-4. Here things are just as bad, but in the opposite direction: several of the frames are empty, and only four percent of the space allocated to each group is being used—a very inefficient use of disk space.

```
FILE= MD MODULO= 79              15:54:46  01 NOV 1989
FRAMES BYTES ITMS
     1    119   5 *>>>>>
     1    178   9 *>>>>>>>>>>
     1     34   2 *>>
     1    119   6 *>>>>>>
     1     24   1 *>
     1      0   0 *
     1     79   4 *>>>>
         .
         .
         .
     1     62   2 *>>
     1    117   5 *>>>>>
     1    105   6 *>>>>>>
     1     36   2 *>>
    79

ITEM COUNT=      280, BYTE COUNT=   6010, AVG. BYTES/ITEM=   21.4
AVG.ITEMS/GROUP= 3.5, STD.DEVIATION= 2.0, AVG. BYTES/GROUP=  76.0
```

Example 6-4.
HASH-TEST showing a poor modulo choice.

Copying Items to and from Tape

The T-DUMP and S-DUMP verbs copy file items to magnetic tape. The T-LOAD verb copies items from magnetic tape back to disk. These verbs can be used to maintain copies of entire files or selected file items off-line. All three of these verbs are used with a tape unit that has been previously attached with the T-ATT verb.

Copying Items to Tape

The T-DUMP and S-DUMP verbs create a tape label, copy items to tape, and write an end-of-file (EOF) marker on the tape after the operation is complete. These verbs can copy any file items except File Definition items (D-pointers).

The T-DUMP verb uses the following syntax:

T-DUMP [*file-modifiers*] *filename* [*items*] [*selection*]
[**HEADING** " *text* "] [*modifiers*] [*options*]

The S-DUMP command uses the same syntax as T-DUMP and additionally allows sort expressions. S-DUMP lets you copy items to magnetic tape in sorted order.

The text defined by the HEADING modifier is added to the standard tape label produced during the copy operation. As shown in Table 6-2, there are only three other formatting modifiers that are applicable to the T-DUMP and S-DUMP verbs.

Table 6-2. T-DUMP and S-DUMP Modifiers.

Modifier	Description
HDR-SUPP	Suppresses the creation of a tape label. The H option can also be used.
HEADING	Defines additional text for the tape label.
ID-SUPP	Suppresses the listing of item IDs during the copy operation. The I option can also be used.
SUPP	Same as HDR-SUPP

Any other formatting modifiers used with T-DUMP, S-DUMP, or T-LOAD are ignored.

The statement shown in Example 6-5 copies selected items from the CUSTOMERS file to magnetic tape in sorted order by last name, and creates a tape label that reads "CUSTOMERS A-K 03/22/88". The system automatically lists the item IDs of the copied items on the terminal screen.

Copying Items from Tape

The T-LOAD verb copies items back to disk that were previously copied to magnetic tape with the T-DUMP or S-DUMP verbs. These items can only be copied to an existing file. If you include a selection expression, T-LOAD uses the Attribute Definition items of the dictionary of the file you are loading into, in order to interpret the layout of the items on the tape.

```
>S-DUMP CUSTOMERS WITH LAST-NAME < "LAMPSON" BY LAST-NAME
    HEADING "CUSTOMERS A-K 03/22/88"

 BLOCK SIZE: 16896
    1 MASHX5777
    2 JBOHA5422
    3 JBROW6749
    4 JBUCK6488
    5 DEDGE6635
    6 AEDWA5224
    7 HHIGG6849
    8 HJENK7129
    9 AJOHN5396
   10 HJOHN7265
10 ITEMS DUMPED.
```

Example 6-5.
S-DUMP copies specified items to tape in sorted order.

After the copy operation is complete, the tape is once again positioned just after the end-of-file (EOF) mark.

T-LOAD uses the following syntax:

T-LOAD [*file-modifiers*] *filename* [*items*] [*selection*] [**ID-SUPP**] [**(I)**] [**(O)**]

The I option and the ID-SUPP modifier suppress the listing of item IDs during the copy operation. The O option overwrites the items on disk that correspond to items copied from tape. If you don't include the O option in the query, the system displays an error message for each file item that already exists on disk.

The following statement copies from magnetic tape all items from the ORDERS file whose dates fall in November and December of 1987. This example also overwrites the corresponding file items on disk and suppresses the listing of item IDs.

```
>T-LOAD ORDERS WITH DATE GE "11/01/87" AND LE
  "12/31/87" (I,O)

378 ITEMS LOADED.
```

Tape Format Verbs vs. the TAPE Modifier

The TAPE modifier can be used with verbs such as LIST and SORT to retrieve and list data from a T-DUMP or S-DUMP tape. Generating reports with this modifier differs from using the T-LOAD verb in that TAPE does not copy file items to disk. The TAPE modifier merely specifies that the currently attached tape should be accessed and that the data on this tape should be used to produce the report.

The following query generates a report from the CUSTOMERS file as it is stored on tape:

```
>LIST CUSTOMERS LAST-NAME FIRST-NAME STREET CITY
  STATE ZIP TAPE
```

Restructuring File Items

The REFORMAT and SREFORMAT verbs restructure file items and direct the output to magnetic tape, to another file, or to the original file.

Restructuring file items is useful for rearranging data in file items, for streamlining files by eliminating superfluous data, and for creating cross-reference files.

REFORMAT and SREFORMAT do not copy raw data to another file or to tape; they redirect output. This means that any correlatives and conversions will be applied to the data before it is output. If you do not want correlatives and conversions applied to the data, i.e., you want stored, raw data, create synonyms for the Attribute Definition items that point to the data you want to

you do not enter a starting serial number, 0 is assumed. If you do not want the serial number to be printed on the form, replace the *x*- and *y*-coordinates with (–1).

To create an audit trail:

1. Create the audit file before you enter the forms generation statement. The audit items will be added to this file as the forms are generated.
2. In the forms generation statement, use the @C print code with the attributes that you want to track. The data for these attributes will be stored in the audit file, but will *not* appear on the forms. Even if you track more than one attribute, however, the system generates only one serial number per form.

   If you specify *x*- and *y*-coordinates with a @C-formatted attribute, the serial number is 1) output at the specified locations on the form in place of the data, and 2) entered as an item ID in the audit file. To suppress the display of the serial number on the form and just maintain it in the audit file, enter "–1" in place of the *x*- and *y*-coordinates.

Audit file items have the following format:

Part of Item	Contents	
Item ID	Serial number. .	
Attribute 1	0	If this page is not the last page of the form.
	1	If this page is the last page of the form.
	A	If this page was part of the printer alignment process.
Attribute 2	Current system date in internal format.	
Attribute 3	Current system time in internal format.	
Attribute 4	Data for the first @C-formatted attribute in the forms generation statement.	
Attribute 5	Data for the second @C-formatted attribute in the forms generation statement.	
Attribute *n* + 3	Data for the *n*th @C-formatted attribute in the forms generation statement.	

If a serial number already exists as an item in the audit file, the system overwrites the old item with the new one. Otherwise, new items are added to the file.

reformat, and leave out the correlatives and conversions. Then use the synonyms in the REFORMAT or SREFORMAT statement.

To reformat an attribute that uses a correlative to derive its data from another file, create a synonym for the Attribute Definition item that includes the correlative (otherwise there will be no data) but does not include any conversions.

After you enter the REFORMAT or SREFORMAT statement, the system displays the following prompt:

```
FILE NAME:
```

At this point, you can perform one of three different operations:

1. Transfer items to magnetic tape by entering the word TAPE. You must already have attached the tape unit and set the tape record length with the T-ATT verb.
2. Transfer items to another file by entering the name of an existing file as follows:

   [**DICT**] *filename*

3. Transfer items within the same file by pressing the RETURN key.

The following sections contain detailed information about each of these operations.

Transferring Items to Tape

To copy items to magnetic tape, enter the word TAPE in response to the FILE NAME: prompt. A tape label that includes the block size, time, date, and filename is written to the tape, then the data from the specified attributes is dumped in the order specified by the REFORMAT or SREFORMAT statement. One tape record is created for each item specified.

Reformatted items are not written to tape in T-DUMP format, so they cannot be accessed by other ACCESS queries using the TAPE modifier. They can, however, be read using T-READ or by a PICK/BASIC program.

Nor is it possible to specify your own tape label with the HEADER modifier as you can when you write items to tape using T-DUMP and S-DUMP. You can, however, suppress the default tape label with the HDR-SUPP modifier.

Before you write items to tape you should use T-ATT to specify a block size for tape records proportional to the amount of data you are going to write. For example, if the data you want to write amounts to less than 100 characters for each item, it makes sense to specify a block size of 512 rather than a larger block size, such as 16,896.

Restructuring Items in a New File

You can copy all of the data in the original file to the destination file, or you can specify only those attributes whose data you want copied. This means that the destination file can contain a subset of the attributes in the original file.

You can also specify a different order for the attributes in the destination file. Furthermore, the data in any attribute of the original file can be specified as the item IDs in the destination file.

Here are two possible reasons for restructuring file items in this way:

1. You no longer need to maintain all of the information in the original file items on-line. To conserve disk space, you restructure the items by eliminating the attributes that have become superfluous and thereby shrink the file. This is much more efficient than individually editing the file items in the original file with the Editor.
2. You want the attributes in the original data file items to be ordered differently. The attribute numbers in the file dictionary can be edited to reflect the new sequence, but changing the order of data in each file item would be a more involved task. Reordering the data of every file item can be accomplished with a single REFORMAT statement.

Only the attributes you specify as output in the REFORMAT statement are included in the destination file. Furthermore, if you include the ID-SUPP modifier in the statement, the first attribute listed as output becomes the item ID in the destination file.

What About File Dictionaries?

File dictionaries are not affected by the restructuring operations performed on data file items. This means that you must edit the Attribute Definition items in the file dictionary so that they reflect the new item structure. This can involve changing:

- The attribute number (AMC) to reflect the new order in which information is stored in the data file.
- Any correlatives that reference other attributes in the file (for example, the T*file* correlative).
- The parameters for Controlling and Dependent attributes.
- Any PICK/BASIC programs that access the file (and recompiling them).

If you are directing restructured items to another file, you must either copy the corresponding Attribute Definition items from the original file dictionary to the destination file dictionary with the COPY verb and then edit them, or you must create new Attribute Definition items. If you do not edit the affected Attribute Definition items, it will be impossible to properly access the newly-structured data when you try to generate an ACCESS report. Editing a file dictionary in this way is illustrated in the next section.

Example

Let's assume that you want to restructure the CUSTOMERS file and use the customers' social security numbers (currently Attribute 8, SS#) as item IDs instead of the 10-character customer IDs. You also plan to eliminate Attribute 9, COUNTY, in each data file item, but retain Attribute 10, SEX. Finally, you plan to split the CUSTOMERS file into two separate files: one file called CUSTOMERS.INDIV for private individuals and the other called CUSTOMERS.CORP for corporations. You can do this by referencing Attribute 11, CUST.TYPE, with a selection expression.

Restructure the CUSTOMERS file in this way by doing the following:

- Create two new files with the names CUSTOMERS.INDIV and CUSTOMERS.CORP:

```
>CREATE-FILE CUSTOMERS.INDIV 7 57

[ 417 ] FILE 'CUSTOMERS.INDIV' CREATED; BASE 8276,
   MODULO 7

>CREATE-FILE CUSTOMERS.CORP 7 57

[ 417 ] FILE 'CUSTOMERS.CORP' CREATED; BASE 9485,
   MODULO 7
```

- Transfer restructured data file items to these files with two separate REFORMAT statements. The attribute CUST.TYPE in the original CUSTOMERS file is used in a selection expression to separate the data file items into the two new categories:

```
>REFORMAT CUSTOMERS WITH CUST.TYPE = "I" SS#
   LAST-NAME FIRST-NAME STREET CITY STATE ZIP
   PHONE SEX ID-SUPP

FILE NAME: CUSTOMERS.INDIV

>REFORMAT CUSTOMERS WITH CUST.TYPE = "C" SS#
   LAST-NAME FIRST-NAME STREET CITY STATE ZIP
   PHONE SEX ID-SUPP

FILE NAME: CUSTOMERS.CORP
```

  The ID-SUPP modifier is included in these statements. This means that SS#, the first attribute in the output specification, becomes the new item ID for both of the new files.

  The rest of the output specification reflects the new structure of the data items: the COUNTY attribute is not included, nor is the CUST.TYPE attribute since it is now superfluous.

- Copy the necessary Attribute Definition items from the CUSTOMERS dictionary to the two new dictionaries:

```
>COPY DICT CUSTOMERS 'FIRST-NAME' 'LAST-NAME'
   'STREET' 'CITY' 'STATE' 'ZIP' 'PHONE' 'SEX'
TO :(DICT CUSTOMERS.INDIV
   .
   .
   .
>COPY DICT CUSTOMERS 'FIRST-NAME' 'LAST-NAME'
'STREET' 'CITY' 'STATE' 'ZIP' 'PHONE' 'SEX'
TO :(DICT CUSTOMERS.CORP
   .
   .
   .
```

- Edit these Attribute Definition items so that the attribute numbers reflect the new item structure.

Table 6-3 compares the old and new attribute numbers.

Table 6-3. Attribute Numbers Before and After Reformatting.

Attribute Name	Old Number	New Number
FIRST-NAME	1	2
LAST-NAME	2	1
STREET	3	3
CITY	4	4
STATE	5	5
ZIP	6	6
PHONE	7	7
SS#	8	Item ID
COUNTY	9	Not used
SEX	10	8
CUST.TYPE	11	Not used

CHAPTER 7

Forms Generation

Queries that generate reports to be printed on forms such as invoices, checks, order forms, etc., are called *forms generation* statements. These statements allow you to create the forms themselves as well as to output data on preprinted stock. Some versions of the Pick system, such as ADDS Mentor, provide special ACCESS verbs (FORMS and SFORMS) that allow you to output data on preprinted forms. Other versions, such as Ultimate, offer the same capability but include it as extensions to the LIST and SORT verbs.

Pick systems that offer the forms generation capabilities described in this chapter make use of an enhancement of ACCESS called the English Forms Generator, also known as EFoG. Although the implementation of EFoG differs from system to system, the features described in this chapter work generally as they are documented here.

This chapter contains the following sections:

- **Forms, Items, Pages, and Subpages**: Defines the relationship between these elements when generating both single- and multipage forms.
- **Forms Generation Statements**: Describes the syntax of forms generation statements.
- **Generating Forms: An Overview**: Summarizes the use of print codes. This section also describes how to output data on simple forms and how to use modifiers in forms generation statements.

- **How Data Appears on Forms**: Discusses how Attribute Definition items affect the way data appears on forms. This information helps avoid unwanted truncation or overlapping.
- **Including Multivalued Attributes on Forms**: Discusses the effects of including multivalued attributes in forms generation statements. Specifically, this section gives instructions on how to design single- and double-depth windows, how to print data on the first and last pages of a multipage form, and how to number pages.
- **Special Features of Forms Generation**: Describes the procedures for creating a background form, aligning the printer, and maintaining an audit trail.

Forms, Items, Pages, and Subpages

The relationship between forms, items, pages, and subpages should be understood before you begin working with forms generation statements. A typical ACCESS report displays a group of data items in columnar or linear format. A *form*, on the other hand, displays individual items, one per form, in a custom-designed layout.

A form can be thought of as a template for the format of a single item of data. For example, you might generate invoices from an ORDERS file, producing one invoice form for each order item. A forms generation statement allows you to select the items to be output and to position the data properly on the invoice forms.

A *page* is a piece of paper (if the report is sent to the printer) or a screenful of data (if the report is displayed at the terminal) whose length and width is set by the TERM verb. The relationship between a page and a form varies, according to whether the report includes multivalued attributes and whether more than one item is to be printed or displayed per page.

The FORMS and SFORMS verbs position only one item on each separate page. If a report includes multivalued attributes, all of the data for an item might not fit on one page. In this case, data spills onto the next page, creating a multipage form. Working with multipage forms involves a number of

additional considerations. This topic is covered fully in the later section "Including Multivalued Attributes on Forms."

It is also possible to include more than one item on a page. Multiple items on a single page are referred to as *subpages*. Subpages are equal-sized sections of a page made up of the number of lines you specify. For example, a 60-line page might have six 10-line subpages. Each subpage (item) is treated as a separate form. In this way a single physical page can hold multiple forms.

Subpages are implemented differently on different systems. Some systems, such as ADDS Mentor, have two ACCESS verbs, REPT and SREPT that position more than one item on a page. Other systems, such as Ultimate, use the M option to create subpages. The M option is described in the section, "Forms Generation Options," later in this chapter.

Forms Generation Verbs

Some systems (ADDS Mentor is one) use special forms generation verbs, such as FORMS and SFORMS, to create forms generation statements. Other systems, such as Ultimate, incorporate forms generation syntax into the verbs LIST and SORT. The syntax is similar (though not identical) for both implementations.

Table 7-1 summarizes four ACCESS verbs used to position and output data on forms.

Table 7-1. Forms Generation Verbs.

Verb	Description
FORMS	Outputs items on pages.
SFORMS	Outputs items on pages in sorted order.
REPT	Outputs items on subpages.
SREPT	Outputs items on subpages in sorted order.

Forms generation verbs use the following general syntax:

verb [**DICT**] *filename* [*items*] [*selection*] [*sort*] *output* [*print*]
[*modifiers*] [(*options*)]

Most of these parameters have been described fully in Chapter 2.

The *output* parameter specifies the attributes whose data is to be output on the form. Each attribute must be associated with a *print code*, as described in the "Using Print Codes" section of this chapter. At least one output specification must be included in a forms generation statement, which means that the default output specifications do not apply.

Certain ACCESS *modifiers* behave somewhat differently when used with form generation verbs. These modifiers are described in the "Using Modifiers in Forms Generation Statements" section of this chapter. There are also special options designed to work with forms generation verbs. They are described later in this chapter, in the section, "Forms Generation Options."

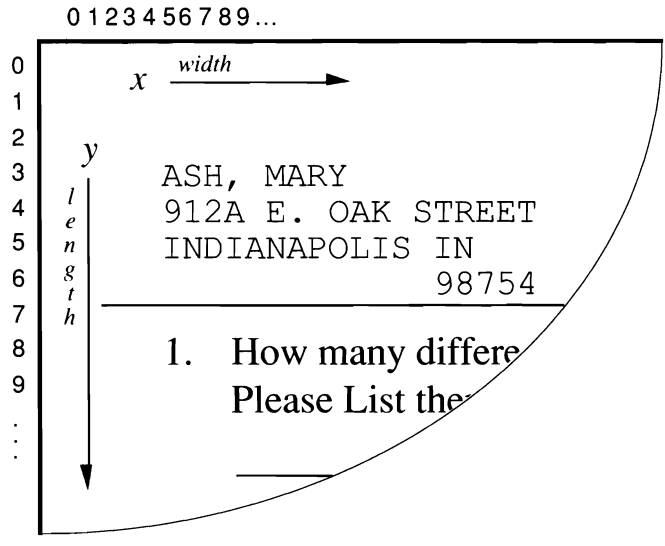
Generating Forms: an Overview

Forms generation verbs provide powerful formatting capabilities for ACCESS reports. These verbs make it possible to design a form and to specify how data should appear on it. For example, you can design invoices, checks, and shipping forms.

A form can be created in any of the following three ways: by specifying and positioning text to create a form template, while simultaneously positioning data on it; by designing a background form separately and identifying it in the forms generation statement (background forms are described later in this chapter); or by using a preprinted paper form and placing data on it.

Using Print Codes

The output specifications in the forms generation statement associate each attribute with a *print code* that explicitly positions the data on the form. The placement of data is specified by *x*- (column) and *y*- (row) coordinates. The upper left corner of the form is the origin of the coordinate system (0,0)—see Example 7-1.



Example 7-1. Positioning Data.
Data is positioned on a form by specifying x- and y-coordinates.

The simplest output specification in a forms generation statement takes the following form:

@(*x*,*y*):*attribute-name*

x	is the horizontal position (column) on the page where the data begins. The leftmost position is column 0.
y	is the vertical position (row) on the page where the data begins. The top row is row 0, which is reserved for the heading.
attribute-name	is the name of the attribute whose data is printed at the specified position.

This specification simply defines a position on the form where the data for a given attribute will be output. For example, the specification @(2,5):LAST-NAME outputs a last name on a form, beginning at column 2 on row 5 (see Example 7-1).

More complex output specifications can be defined with special print codes that take the following form:

code : *attribute-name* [**1**,*n*]

In addition, a character string that prints in a specified location on the form can be defined with the following syntax:

code : "*string* "

The parameters for both syntaxes are shown below:

code	is the @ code associated with the attribute. The available print codes are shown in Table 7-2.
attribute-name	is the name of the attribute whose data is printed at the specified position.
string	is a character string that is printed at the specified position.
n	prints only the first *n* characters of the data for this attribute.

The system does not check whether output data exceeds the page width set by the TERM verb. The system uses the specified attribute's justification (line 9) in combination with the column width (line 10) and the column position specified by *x* to set the display width of the data. For more information about how justification and column width effect the display of data on forms, see the section, "How Data Appears on Forms," later in this chapter.

Table 7-2 summarizes the print codes.

Table 7-2. Print Codes for Forms Generation Verbs.

Code	Description
@[A] (*x*,*y*)	Prints the data for an attribute on every page of a form at the location *x*, *y*.
@C(*x*,*y*)	Creates an audit trail for a series of forms. This code can also be used to serialize the forms. To suppress serialization numbers on the form (but not in the audit file), specify (–1) in place of the *x*- and *y*-coordinates.

@D(*x*,*y*,*z*)	Prints data for multivalued attributes in double-depth windows. This makes it possible to define two lines of output at a time. *z* specifies the bottom-most row of a window whose top-most row is defined by *y*. An "S" must be added to the end of the *x* parameter for an attribute to appear on every second output line.
@F(*x*,*y*)	Prints the data for an attribute on only the first page of a multipage form.
@L(*x*,*y*)	Prints the data for an attribute on only the last page of a multipage form.
@M(*x*,*y*,"*text*")	Prints the data on a single-page form. On a multipage form it prints the specified text on all but the last page of a multipage form, then prints data on the last page of the form.
@W(*x*,*y*,*z*)	Prints data for multivalued attributes in windows. *z* specifies the bottom-most row of a window whose top-most row is defined by *y*.

Designing a Form

As described earlier, the simplest output specification in a forms generation statement assumes the A print code as the default, and takes the following form:

@(*x*,*y*) : *attribute-name*

This specification simply defines a coordinate on the form where the data for a specified attribute will be output. 0,0 is the upper-left corner of the page. The maximum value for *x* is the width of the page and the maximum value for *y* is the length of the page.

Let's assume you want to generate a questionnaire form for each customer in the CUSTOMERS file. A one-page questionnaire has already been created on preprinted stock. The forms generation statement should output each customer's name and address in the blank space provided at the top of each page.

The customer's full name is printed on the third line of each form, the street address on the fourth line, the city and state on the fifth line, and the zip code on the sixth line. This block of information begins at the fifth column

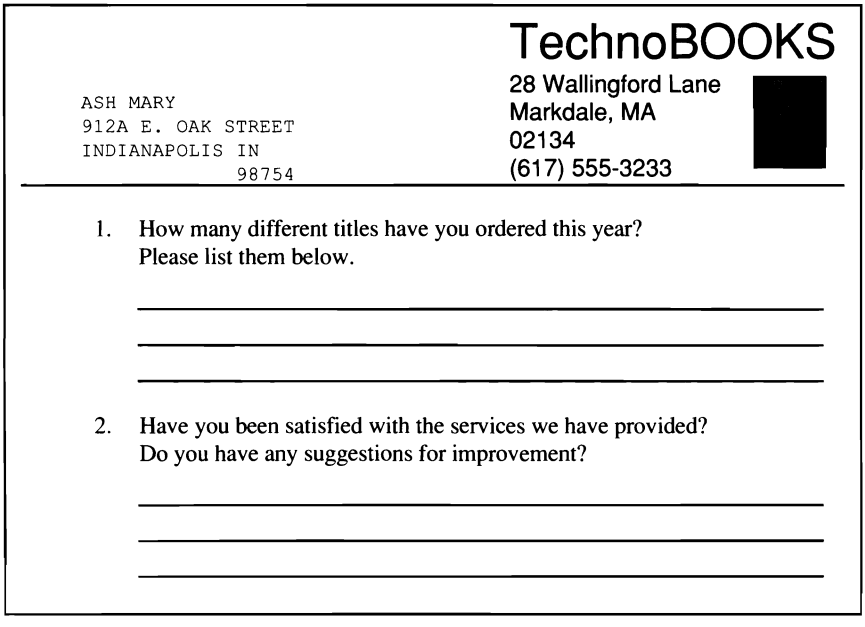
position in each of these four rows; the state and zip code both begin at the eighteenth column position. This means that there are ten available character spaces for the city. (These values correspond to the column width defined by line 10 of the Attribute Definition item CITY.) The default heading is suppressed, so page numbers, the current time, and the date do not appear on the forms.

The preprinted stock is loaded into the printer tray and the following statement is entered:

```
>SFORMS CUSTOMERS BY LAST-NAME @(5,3):FULL-NAME
 @(5,4):STREET @(5,5):CITY @(18,5):STATE @(18,6):ZIP
 HDR-SUPP
```

The SFORMS verb generates the questionnaires in sorted order, by customers' last names.

Example 7-2 shows how the customer's name and address is output at the top of the form.



Example 7-2.
The SFORMS verb prints data in the positions specified by print codes. The forms are printed in sorted order.

Later sections of this chapter describe more complex types of forms generation that involve multivalued attributes, multipage forms, background forms, and audit trails.

Using Modifiers in Forms Generation Statements

Certain ACCESS modifiers behave differently when used with forms generation verbs. Table 7-3 summarizes these differences.

Table 7-3. Forms Generation Modifiers.

Modifier	Description
BREAK-ON	Data line is treated like any output specification for the purposes of formatted printing.
FOOTING	Data cannot be output with *x*- and *y*-coordinates in the rows defined for the footing.
GRAND-TOTAL	Data line is treated like any output specification for the purposes of formatted printing.
HDR-SUPP	Data cannot be output with *x*- and *y*-coordinates in row 0.
HEADING	Data cannot be output with *x*- and *y*-coordinates in the rows defined for the heading.
ID-SUPP	Not necessary in forms generation statements. Item IDs are not automatically included on forms.
TOTAL	Data line is treated like any output specification for the purposes of formatted printing.
WINDOW	A special modifier on Ultimate systems that lists multivalued attributes on forms.

The following sections discuss these modifiers in more detail.

BREAK-ON, TOTAL, and GRAND-TOTAL Modifiers

Generally, forms will not use the BREAK-ON, TOTAL, and GRAND-TOTAL modifiers. There are two situations, however, where they can be useful.

First, you can generate a total at the end of a run of forms by totalling the attributes you want to display or print. The BREAK-ON modifier is not needed for output. If an audit file is being generated and the C-coded items have total modifiers on them, the highest-numbered item ID in the audit file will contain the grand totals.

Second, you can force a control break at specified locations. For example, by using the NI function in an A or F correlative in an attribute with a BREAK-ON modifier, you can generate and output a total after every *n* forms.

Including Item IDs

Unlike most other ACCESS verbs, forms generation verbs do not automatically include item IDs in reports. Therefore you need not use the ID-SUPP modifier to suppress this display.

If you *do* want item IDs to appear on forms, you should create an Attribute Definition item in the file dictionary for the item ID that specifies 0 as the attribute number. Include an output specification for this attribute in the forms generation statement.

Defining Headings and Footings

The four forms generation verbs automatically produce the standard ACCESS report heading on every page. This heading contains the current time, date, and page number. Forms generation verbs do not produce a default footing.

The HEADING and FOOTING modifiers can be included in a forms generation statement to define customized headings and footings, just as in other ACCESS queries. When working with forms, however, keep the following in mind:

- The system automatically reserves space at the top of every page for the heading. If you try to output data with *x*- and *y*- coordinates in this reserved space, the system displays an error message.
- If the default heading is used, the system reserves rows 0 and 1: the default heading (time, date, page number) is printed on row 0, and a

blank line, which is considered part of the standard heading, is printed on row 1.

- If the HDR-SUPP modifier is included in the forms generation statement, the system reserves only row 0 and leaves it blank. If you define a heading with the HEADING modifier, the system reserves row 0 plus any additional lines needed for the specified text.
- If you define a footing with the FOOTING modifier, the system reserves the required number of rows at the bottom of the page. Data cannot be output in this area with *x*- and *y*-coordinates.
- The Z option, described later in this chapter, can be used to reset page numbers for multipage forms.

The WINDOW Modifier

On Ultimate systems, the WINDOW modifier is used to implement the listing of multivalues on forms. On other systems this feature is handled by the @W and @D print codes. See the section, "Including Multivalued Attributes on Forms," later in this chapter.

Forms Generation Options

Table 7-4 summarizes the options available to all four forms generation verbs.

Table 7-4. Options for Forms Generation Verbs.

Option	Description
A	Runs a top-of-form printer-alignment routine. See the section, "Aligning the Printer," later in this chapter.
B	Includes previously-created text or graphics from a background form on every page of a form. See the section, "Creating a Background Form," later in this chapter.
M	Used by some systems (i.e., those that do not support the REPT and SREPT verbs) to specify the size of a subpage. Subpages allow you to print more than one item per page.

When the M option is used, the system prompts you to enter the number of lines per subpage.

Z — When used with the FOOTING modifier, resets the page number in the footing back to 1 at the beginning of each multipage form. See the section, "Numbering Pages on Multipage Forms," later in this chapter.

How Data Appears on Forms

This section contains guidelines to help you avoid unwanted truncation or overlapping of data on forms.

When you are designing a form, keep the following in mind:

- The limits of a page's usable width and length are defined by the TERM verb.
- The justification and column width in the Attribute Definition items for each attribute affect the way data appears on forms.
- Null-value Controlling attributes can affect the appearance of the data.
- Null-value attributes translated from another file with the T*file* correlative can affect the appearance of the data.
- The system reserves space for the default heading, or for headings and footings defined as part of the forms generation statement. The use of headings and footings is described earlier in this chapter.

You can check the positioning of the data on a set of forms by using the A option, as described in the section, "Aligning the Printer."

Coordinating Justification and Column Width Fields

When you position data on a form with *x*- and *y*-coordinates, the system also uses the justification and maximum column width defined for each attribute for placement and truncation. Table 7-5 shows how this can affect the appearance of data on a form.

Table 7-5. Justification of Data on Forms.

Justification	Result
L	Data is left-justified. Output begins at the specified column number (x) and extends through column (x + column width). Data that exceeds the length set by the column width field is truncated at column (x + column width), the rightmost position.
R	Data is right-justified. Output begins at the specified column number (x) and extends through column (x + column width). Data that exceeds the length set by the column width field is truncated at column x, the leftmost position.
T	Data is left-justified. Output and truncation work the same way as data whose justification is L, except if data is output in a window. In this case, data wraps to the next line on a word boundary.

Using Controlling and Dependent Attributes

If a Controlling attribute has no data, ACCESS ignores any associated Dependent attributes. This can cause problems in forms generation, however, by displacing the rest of the data on a form. There are two possible remedies for this situation. First, you can create synonyms and remove the controlling and dependent specifications from the Attribute Definition items. These specifications are not necessary unless you plan to use print limiters in the forms generation statement.

Another remedy is to place the Controlling and Dependent attributes at the end of the output specifications for the set of forms. Putting these attributes at the end prevents them from displacing the rest of the data. This remedy, however, allows you to include only one set of Controlling and Dependent attributes in each forms generation statement.

Using the T*file* Correlative

If a T*file* correlative returns a null value, the output specification for that attribute is ignored. This can cause subsequent data to print or display incorrectly on the form. As a preventive measure, do one of the following when using the T*file* correlative:

1. Move the T*file* correlative from line 8 to line 7.
2. Use a C or a V action code in the T*file* correlative instead of an X action code. Both of these action codes prevent a null value from being returned if the conversion fails.
3. Embed the T*file* correlative within an arithmetic correlative (A or F) and append a single blank to it. This ensures that the attribute contains at least one blank and will therefore not be considered null. For example, if Attribute 3 is used as the data source, the following T*file* correlative:

   ```
   TORDERS ; X ; ; 10
   ```

   could be replaced by this one:

   ```
   A ; 3(TORDERS ; X ; ; 10):" "
   ```

For more information about the T*file* correlative, see Chapter 8.

Including Multivalues on Forms

Forms that do not contain multivalued attributes are relatively easy to design. These forms each occupy a single physical page and there is a one-to-one correspondence between each output specification and the resulting data element.

Special care must be taken when building a forms generation statement that prints one or more multivalued attributes on the form. This is because multivalued attributes have special formatting requirements and sometimes cause the data to occupy multiple pages on a form.

When working with multivalued attributes, you need to be aware of the following considerations. First, window specifications (using the @W or @D print codes) for these attributes must be included in the forms generation statement so that the data can be properly displayed. Second, you will need to

use the @F, @L, and @M print codes to control the display of text and data on multipage forms. Finally, you can set special page numbering for multipage forms with the Z option.

As was mentioned earlier, Ultimate systems implement the listing of multivalues on forms with the WINDOW and END-WINDOW keywords; the @W and @D print codes are not used. The syntax of WINDOW is:

WINDOW @(*x,y,z* [{ 1 | 2 }]) [: "*string*" : @(*n*) : "*string*" ...]]

where *x*, *y*, and *z* specify the left-most column, top row and bottom row of the window. "1" and "2" specify whether each multivalue should occupy one or two lines. A colon indicates concatenation. *n* specifies the column position on the current line where *string* is to be printed or displayed.

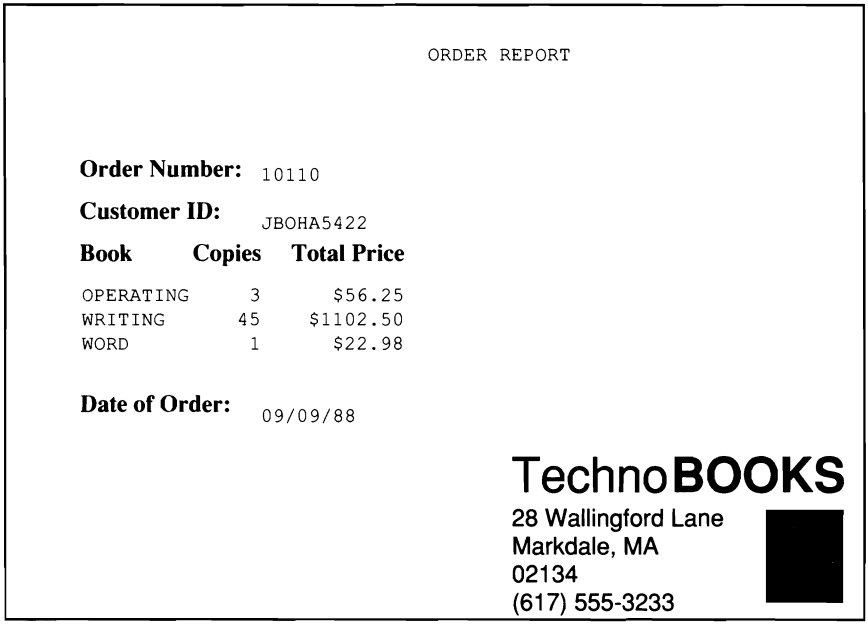
A WINDOW phrase in a forms generation statement must always be followed by the END-WINDOW modifier.

The following sections describe how to treat multivalued attributes when working with forms.

Designing Windows

Data in multivalued attributes must be output in *windows*. A window is an area on a form whose position and depth you define with print codes or with the WINDOW modifier. A single form can contain up to six different windows.

A window can contain either one or two lines for each set of associated values in a multivalued attribute. A one-line window is called a *single-depth* window, a two-line window is called a *double-depth* window. Single-depth windows are defined with the @W print code, double-depth windows are defined with the @D print code, except on Ultimate systems, where single or double depth windows are specified by a "1" or a "2" in the WINDOW syntax line.



Example 7-3.
First page of a form with single-depth windows.

Single-Depth Windows

A window can contain any number of multivalued attributes. For example, the following forms generation statement defines a window that contains the multivalued attributes SHORT.TITLE, QTY, and LINE.AMT:

```
>FORMS ORDERS @(20,5):ORDER.ID @(20,7):CUST.ID
  @W(5,10,12):SHORT.TITLE @W(17,10,12):QTY
  @W(22,10,12):LINE.AMT @(20,15):DATE HEADING " 'C'
  ORDER REPORT"
```

These specifications define a window that begins in column 5 and row 10, and extends down the page to row 12, a depth of three lines. This means that if any of the three referenced attributes contains more than three different values, the window continues on the next page of the form.

Each form also includes the item ID for the ORDERS file on the fifth row of the form, the customer ID on the seventh row, and the order date on the fifteenth row. These three attributes all have a single data element per item; in other words, they are not multivalued.

Example 7-3 shows the first page of a two-page form generated by the preceding forms generation statement. Example 7-4 shows the second page of the same form. These forms are generated on preprinted stock that contains bold literal text and column headings (Order Number, Customer ID, Date of Order, etc.).

```
                                ORDER REPORT


Order Number:  10110
Customer ID:   JBOHA5422
Book        Copies   Total Price
DATABASE       12       $119.40


Date of Order:  09/09/88
```

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

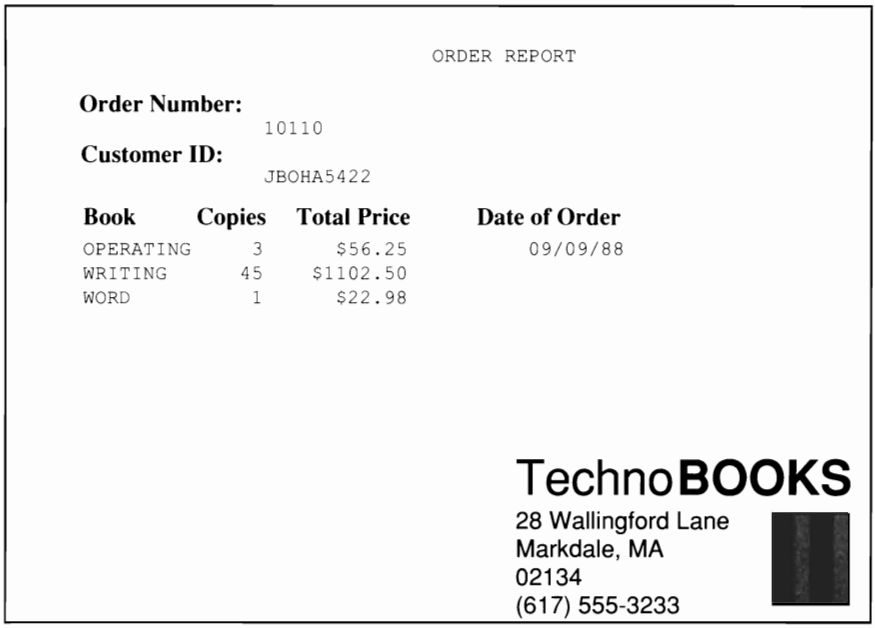
Example 7-4.
Second page of a form with single-depth windows.

The forms also include a centered heading line that reads "ORDER REPORT".

Unused portions of a window can be used to contain additional attributes that are output with some other print code. For example, the following query prints a single-value attribute, DATE, in the unused right-hand columns of the preceding window.

```
>FORMS ORDERS @(20,5):ORDER.ID @(20,7):CUST.ID
  @W(5,10,12):SHORT.TITLE @W(17,10,12):QTY
  @W(22,10,12):LINE.AMT @(42,10):DATE HEADING " 'C'
  ORDER REPORT"
```

The A print code (remember, the A is optional) causes the date of the order to be displayed starting at column 42 on the first line of the window. This data will appear on every page of each form. Example 7-5 shows the first page of the edited multipage form.



ORDER REPORT

Order Number:
10110

Customer ID:
JBOHA5422

Book	Copies	Total Price	Date of Order
OPERATING	3	$56.25	09/09/88
WRITING	45	$1102.50	
WORD	1	$22.98	

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-5. *Data can overlap unused portions of a window.*

If the data for a multivalued attribute exceeds its allotted area in a window and the attribute's justification is "T", the data wraps to the next line of the window and pushes the next set of values down to the subsequent line.

Double-Depth Windows

Double-depth windows reserve two lines of the window for each set of values in a multivalued attribute. For example, since book titles (or any product description) can be very long, you might want to display the data for QTY

and LINE.AMT on one line, and the data for TITLE on the next line. To do so, use the @D print code to specify a position for each of the multivalued attributes, and append an "S" to the column position of the data that is to appear on the second line of each group of two lines (in this case, TITLE). (On Ultimate systems, specify "2" after the *x*-, *y*-, and *z*-coordinates of the WINDOW modifier.)

The following query defines a four-line window. Since the data associated with a set of values takes up two lines, the data for two different titles can be output on each page.

```
>FORMS ORDERS @(20,5):ORDER.ID @(20,7):CUST.ID
 @D(5,10,13):QTY @D(15,10,13):LINE.AMT
 @D(12S,10,13):TITLE @(42,10):DATE HEADING " 'C'
 ORDER REPORT"
```

Example 7-6 shows the first page of a form generated by the preceding statement.

```
                         ORDER REPORT


Order Number:  10110

Customer ID:   JBOHA5422

Copies      Total Price          Date of Order
   3           $56.25                  09/09/88
Book:  OPERATING SYSTEM CONCEPTS
  45         $1102.50
Book:  WRITING COMMERCIAL APPLICATIONS
```

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-6.
First page of a form with a double-depth window.

Printing Data on First and Last Pages of a Form

When a forms generation statement includes data from multivalued attributes, the forms can occupy more than a single page. In this case, use the @F, @L, and @M print codes to customize the forms.

If a forms generation statement references one or more multivalued attributes, the @F code outputs data on only the first page of each multipage form, and the @L code outputs data on only the last page. The remaining pages contain a blank space at the position defined for the @F- or @L-formatted attribute. If each form occupies only a single page (that is, if the data for the multivalued attributes fits in the defined window space), the @F and @L codes function just like the @A code, displaying the data at the specified position on each form. If a form contains no multivalued attributes (and therefore occupies only one page), the @F and @L codes function like the @A code.

ORDER REPORT

MARY ASH
912A E. OAK STREET
INDIANAPOLIS IN

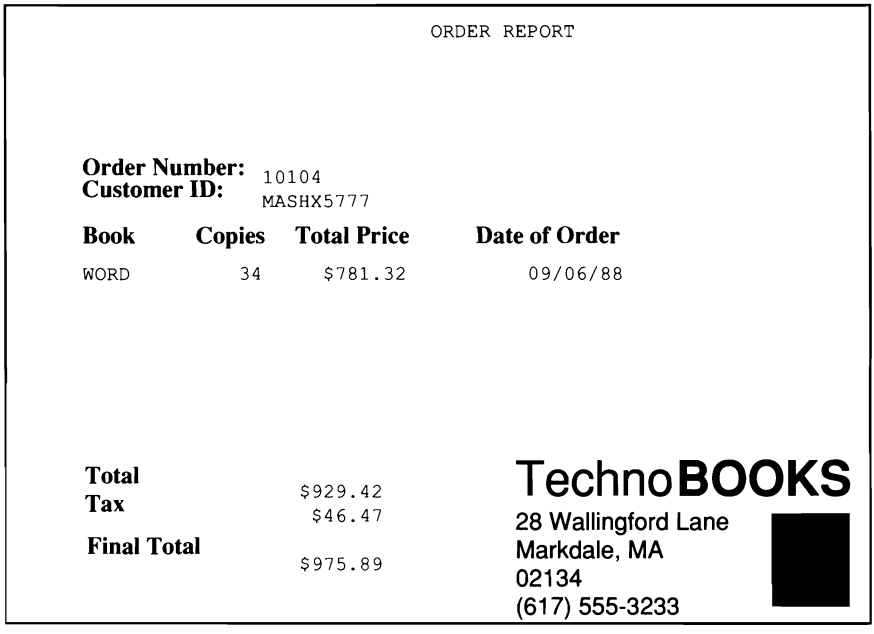
Order Number: 10104
Customer ID: MASHX5777

Book	Copies	Total Price	Date of Order
DATABASE	3	$29.85	09/06/88
OPERATING	5	$93.75	
WRITING	1	$24.50	

Total
Tax
Final Total SEE LAST PAGE

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-7.
First page of a multipage form.



Example 7-8.
Last page of a multipage form.

The @M code outputs text on all but the last page of a multipage form, and then outputs data on the last page. Use the @M code when you want to print a continuation message (for example, "SEE LAST PAGE") on all but the last page of a form and then print the data (such as a total) on the final page. Like @F and @L, the @M code behaves like an @A code if a form occupies only one page: any text is ignored and the data is printed at the specified position.

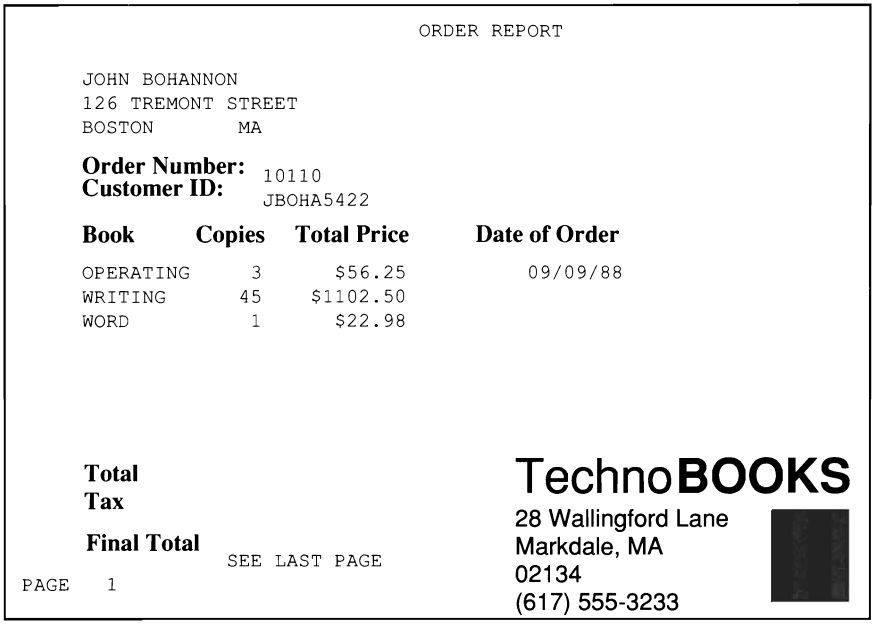
The following statement shows the use of the @F, @L, and @M codes:

```
>FORMS ORDERS @F(5,2):FULL-NAME @F(5,3):STREET
  @F(5,4):CITY @F(18,4):STATE @(20,6):ORDER.ID
  @(20,7):CUST.ID @W(5,10,12):SHORT.TITLE
  @W(17,10,12):QTY @W(22,10,12):LINE.AMT
  @(42,10):DATE @L(20,19):AMOUNT @L(20,20):TAX
  @M(20,22,"SEE LAST PAGE"):TOTAL.AMT HEADING " 'C'
  ORDER REPORT"
```

Example 7-7 shows the first page of a multipage form created with the preceding forms generation statement. Example 7-8 shows the second page of the same form.

Numbering Pages on Multipage Forms

On single-page forms, page numbering is consecutive: the first page (form) is Page 1, the second is Page 2, and so on. By default, page numbering on multipage forms works the same way: all pages produced by a forms generation statement are numbered consecutively. When working with multipage forms, however, you might prefer to number each form separately. You can do so by including the Z option in a forms generation statement.



ORDER REPORT

JOHN BOHANNON
126 TREMONT STREET
BOSTON MA

Order Number: 10110
Customer ID: JBOHA5422

Book	Copies	Total Price	Date of Order
OPERATING	3	$56.25	09/09/88
WRITING	45	$1102.50	
WORD	1	$22.98	

Total
Tax
Final Total

SEE LAST PAGE

PAGE 1

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-9.
The Z option resets the page number to 1 on the first page of each multipage form.

The Z option causes the system to reset the page number in a footing back to 1 on the first page of each multipage form. The Z option is used in combination with the FOOTING modifier. For example:

```
>FORMS ORDERS @F(5,2):FULL-NAME @F(5,3):STREET
   @F(5,4):CITY @F(8,4):STATE @(20,6):ORDER.ID
   @(20,7):CUST.ID @W(5,10,12):SHORT.TITLE
   @W(17,10,12):QTY @W(22,10,12):LINE.AMT
   @(42,10):DATE @L(20,19):TOTAL.AMT @L(20,20):TAX
   @M(20,22,"SEE LAST PAGE"):TOTAL.ORDER HEADING "
   'C' ORDER REPORT" FOOTING "PAGE 'P' " (Z)
```

```
                                ORDER REPORT


Order Number:  10110
Customer ID:   JBOHA5422

Book        Copies   Total Price      Date of Order
DATABASE      12       $119.40           09/09/88


Total                  $1301.13
Tax                      $65.05
Final Total            $1366.18
PAGE   2
```

TechnoBOOKS
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-10.
Last page of a multipage form generated with the Z option and the FOOTING modifier.

Since the Z option does not affect page numbers in headings, the HDR-SUPP modifier should also be included in the forms generation statement, or (as in this case) a heading that does not include a page number should be specified with the HEADING modifier. This prevents two incompatible page numbers from appearing on each page of a form.

Special Features of Forms Generation

The following sections describe the following advanced or special features available for generating forms:

- Creating a background form.
- Aligning the printer to check the positioning of the data.
- Maintaining an audit trail of the generated forms.

Creating a Background Form

As was explained earlier, output from forms generation statements can be printed on preprinted stock. Text can be included in the form itself by using print codes that specify literal strings.

You can also design a *background form* that consists of text or graphics (or both). This background form will be printed on every page of a set of forms along with the data (and any text) specified by the forms generation statement. Background forms make it possible to use standard plain paper to print forms. You can also use printer control characters intended for specific printers for highlighting, underlining, bolding or italicizing, etc.

To use a background form with a set of forms:

1. Create the background form with the Editor. This background form can be stored as an item in any file. Each line of the item corresponds to a line on a page of a form. For example, the third line of the background item is output on the third line of each page of every form.

   Remember to coordinate the placement of the background form with the data and with the headings and footings you plan to use, otherwise portions of the background form or the data could be overwritten.

2. Include the B option in the forms generation statement.

3. Enter the filename and item ID of the background item when the system displays the following prompt:

   ```
   Background File & Item>
   ```

This information should be entered as follows:

filename item-ID

You can also use a Proc or a PICK/BASIC program to specify a background form.

Example 7-11 shows the format of a background item called LATE-NOTICE. The rest of this section demonstrates how this background item can be included on forms:

```
    LATE-NOTICE
001
002
003 YOUR ACCOUNT IS NOW SERIOUSLY OVERDUE
004 IN THE FOLLOWING AMOUNT:
005
006 **********************************
007
008
009
010 **********************************
011
012 PLEASE REMIT PROMPTLY
```

Example 7-11.
The item LATE-NOTICE contains a background form.

The first two lines of the background form are blank, since, at the very least, row 0 is reserved for headings. The fifth and eleventh lines are blank to create white space on the forms, as are the seventh and ninth lines. The eighth line will contain the overdue amount.

The following forms generation statement outputs data on pages that contain the preceding background form:

```
>FORMS CUSTOMERS.OVERDUE @(10,8):AMOUNT-DUE
  @(5,20):FULL-NAME @(5,21):STREET @(5,22):CITY
  @(15,22):STATE HDR-SUPP (B)

Background File & Item> FORM-BACKGROUNDS LATE-NOTICE
```

Example 7-12 shows the resulting output.

```
YOUR ACCOUNT IS NOW SERIOUSLY OVERDUE
IN THE FOLLOWING AMOUNT:

**********************************

          $355.89

**********************************

PLEASE REMIT PROMPTLY


     EDGECOMB DAVID
     338 BROADWAY
     MIAMI    FL
```

TechnoBOOK
28 Wallingford Lane
Markdale, MA
02134
(617) 555-3233

Example 7-12.
This form was generated using the background form shown in Example 7-11.

☞ Printer control characters included in background text can cause data to be displaced from the specified positions. You should test the output with the A option before generating the entire set of forms. This procedure is described in the next section.

Aligning the Printer

The A option lets you manually align the printer before printing an entire set of forms. This is particularly useful when working with a complex forms generation statement that includes, for example, one or more multivalued attributes or a background form.

The A option causes the data on the first form to be (nondestructively) converted to Xs. The converted data is then sent to either the printer or the terminal (depending on whether the forms generation statement includes the P option). The system then displays the following message:

```
Align? Y=cr/N>
```

Inspect the converted data and see whether you need to realign the printer. If you do, manually set the printer at the top of the next form. When this is done, press the RETURN key. The converted data will then be reprinted so you can check it again. If you are satisfied with the converted output and want to print the set of forms, enter "N" and press the RETURN key.

In order to use the A option, your SP-ASSIGNment must include the C and I options. The C option breaks up a print job, sending it to the printer in amounts of 20 frames or less, and the I option specifies that the job be printed immediately.

Maintaining an Audit Trail

Maintaining an audit trail is often desirable when printing checks or generating invoices. The @C print code makes it possible to track the forms created with a forms generation statement by maintaining a separate *audit file* that contains information about each form. This is done by assigning a serial number to each form as it is generated.

An @C print code does two things:

1. Instead of printing the value of the specified attribute on the form at the location specified by the *x*- and *y*-coordinates, it stores the value in the audit file.
2. It prints the serial number on the form at the location specified by the *x*- and *y*-coordinates.

These serial numbers uniquely identify each form and make it possible to track forms individually or as a group. Serial numbers are used as item IDs for items in the audit file.

When a forms generation statement includes a @C code, the system prompts for the name of the audit file and the starting serial number. If you do not enter the name of the audit file, the system will not generate the forms. If

you do not enter a starting serial number, 0 is assumed. If you do not want the serial number to be printed on the form, replace the *x*- and *y*-coordinates with (–1).

To create an audit trail:

1. Create the audit file before you enter the forms generation statement. The audit items will be added to this file as the forms are generated.
2. In the forms generation statement, use the @C print code with the attributes that you want to track. The data for these attributes will be stored in the audit file, but will *not* appear on the forms. Even if you track more than one attribute, however, the system generates only one serial number per form.

   If you specify *x*- and *y*-coordinates with a @C-formatted attribute, the serial number is 1) output at the specified locations on the form in place of the data, and 2) entered as an item ID in the audit file. To suppress the display of the serial number on the form and just maintain it in the audit file, enter "–1" in place of the *x*- and *y*-coordinates.

Audit file items have the following format:

Part of Item	Contents	
Item ID	Serial number.	
Attribute 1	0	If this page is not the last page of the form.
	1	If this page is the last page of the form.
	A	If this page was part of the printer alignment process.
Attribute 2	Current system date in internal format.	
Attribute 3	Current system time in internal format.	
Attribute 4	Data for the first @C-formatted attribute in the forms generation statement.	
Attribute 5	Data for the second @C-formatted attribute in the forms generation statement.	
Attribute *n* + 3	Data for the *n*th @C-formatted attribute in the forms generation statement.	

If a serial number already exists as an item in the audit file, the system overwrites the old item with the new one. Otherwise, new items are added to the file.

Let's assume you want to print employee checks and create an audit trail of the weekly payroll. The audit file items must contain the ID number of the employee receiving the check (ID#) and the amount of the check (NET-PAY). The checks will be serialized according to the numbers automatically generated by the system. These serialization numbers will print on the checks in place of the employee ID#.

You can accomplish all of this with the following forms generation statement:

```
>SFORMS EMPLOYEES BY LAST-NAME @C(9,5):ID#
   @C(-1):NET-PAY @(51,5):DATE @(12,10):FULL-NAME
   @(48,10):NET-PAY @(3,12):NET-PAY-SPELLED (BP)

Background File & Item>CHECK.FORM

Audit File>PAY.AUDIT

Starting Number>4500
```

Notice that the attribute NET-PAY has to be entered twice in the SFORMS statement. The first print code, @C(–1):NET-PAY, is necessary in order to include the amount of each check in the audit file; the (–1) prevents the serial number from being printed a second time on the form. (The serial number will be printed by the print code @C(9,5):ID#). The second print code, @(3,12):NET-PAY, is necessary in order to print the check amount on the form.

The preceding statement prints the checks in ascending order by employees' last names. The data for two attributes is recorded in the audit file (ID# and NET-PAY), but the serialization number appears on the forms only at the location specified for ID#.

The printed checks contain the following information:

- The date.
- The employee's full name.
- The amount of net pay (in numeric characters).
- The amount of net pay spelled out with alphabetic characters.

The forms generation statement also includes the B option and specifies a background form for the checks called CHECK.FORM, and the P option,

which sends the checks to the printer. The audit file is called PAY.AUDIT and the starting serialization number is 4500.

The system then prints the checks and creates the audit file items. A sample audit file item might look like this:

```
     4504
001  1
002  7438
003  180467
004  133326
005  54789
```

This PAY.AUDIT item corresponds to the fifth check printed (#4504). 7438 is the system date and 180467 is the system time, both in internal format. Line 4 contains the employee's ID# (133326), and line 5 contains the amount of the check ($547.89).

The "1" in line 1 indicates that this check was the last check generated for this file item. When a file item generates multipage forms, each page gets a unique serial number with a corresponding audit file item. For example, if a file item generated a three-page form, the audit file items for the first two pages would have a zero in line 1. The third and last audit file item would have a "1" in line 1.

CHAPTER 8

Correlatives and Conversions

Correlatives and conversions are among the most powerful features of ACCESS. They are complex, but once you understand the principles of how they work, you'll find they greatly increase your ability to process and manipulate data both within a file and among different files. Put very simply, correlatives and conversions are processing and output specifications that tell ACCESS what to do with data before listing it in an ACCESS report.

This is a large chapter because of the complexity of the material. The codes are grouped by function, and are arranged in the following sections:

- **An Overview**: Summarizes the functions of correlative and conversion codes.
- **Performing Arithmetic Operations**: Describes the A (algebraic) and F (function) codes, which perform arithmetic and other processing on data contained in or derived from attributes.
- **Deriving Data from Attributes**: Demonstrates the use of correlative codes that manipulate data in one or more attributes. The C (concatenation), G (group extraction), T (text extraction), S (substitution), L (length), P (pattern matching), and R (range) codes are covered.

- **Translating Data from Another File**: Describes the T*file* code, which is used to retrieve data from another file.
- **Formatting Data**: Demonstrates the use of several codes that specify different formats for data before it is output in ACCESS reports. The D (date, MT (masked time), ML and MR (masked decimal), and masked character codes are covered.
- **Advanced Topics**: Explains how and when to place codes in the conversion position (Attribute 7) and in the correlative position (Attribute 8); how to use multiple correlative codes in a single Attribute Definition item; and how to combine correlative and conversion codes.

Appendix C contains reference pages for each of these codes, arranged alphabetically.

If correlative and conversion codes are new to you, you should first read the overview section to get a general idea of what they do and how they work. Then read the next three sections for more details on how to use particular correlative codes, and read the "Formatting Data" section for details on how to use conversion codes.

Throughout Chapter 8 you will find examples of how the codes can be used. If all you want is the syntax for each code, refer to Appendix C.

An Overview

Correlatives and conversions are processing codes which perform special functions on data that is either stored in a file or output in a report. The first section of this overview describes what correlatives are and how they work; some simple examples are included. Next, conversions are explained and illustrated. The third section describes the difference between the way ACCESS handles correlatives and the way it handles conversions.

Correlatives

Correlatives process data that is stored in file items to derive new data. This derived data is not stored in a file item but is newly derived each time the correlative is applied at run time.

For example, a personnel file might store the number of hours each employee works in one attribute, and each employee's hourly rate in another attribute. Hours worked and hourly rate is data held by the system as raw data. Employees' pay, however, need not be stored in the database. Instead, an arithmetic correlative can be used to derive the amount of each employee's pay by multiplying hours worked by the hourly rate. The product of the multiplication is not stored as data in the file but is newly calculated each time the data is processed. This derived data can either be output in a report or be used for further processing.

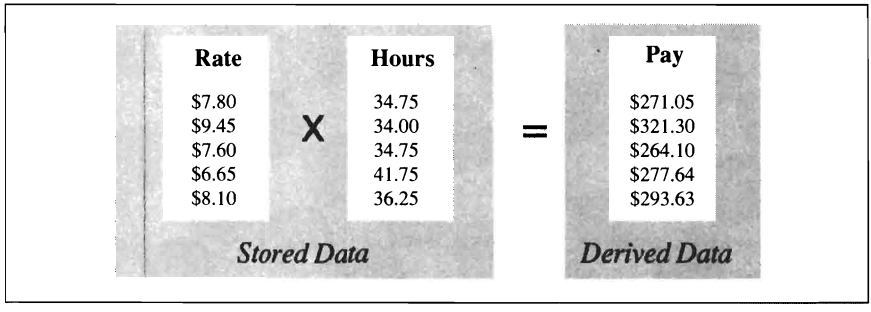


Figure 8-1. A Correlative Processes Stored Data.
The correlative code multiplies Rate by Hours to generate Pay.

Example 8-1 shows what the data derived by this correlative would look like if it were output in a report.

```
>LIST PERSONNEL RATE HOURS PAY HDR-SUPP ID-SUPP

Rate.   Hours    Pay.......

$7.80   34.75    $271.05
$9.45   34.00    $321.30
$7.60   34.75    $264.10
$6.65   41.75    $277.64
$8.10   36.25    $293.63
$5.95   40.00    $238.00
$8.95   42.00    $375.90
$7.90   38.50    $304.15
$8.30   38.75    $321.63
$8.25   39.00    $321.75
$7.50   35.50    $266.25
$8.45   41.00    $346.45
```

Example 8-1.

PAY is derived from the two attributes RATE and HOURS by a correlative.

Arithmetic correlatives such as the one described in the preceding paragraph can be used to perform many mathematical and relational operations on raw data. Arithmetic correlatives are described in the section, "Performing Arithmetic Operations."

Other correlatives can be used to take data from two different attributes and concatenate them together. For example, in a personnel or customer database, last names might be stored in Attribute 1 and first names in Attribute 2. Full names can be derived by a correlative that concatenates them together, with the last name either first or last (see Figure 8-2).

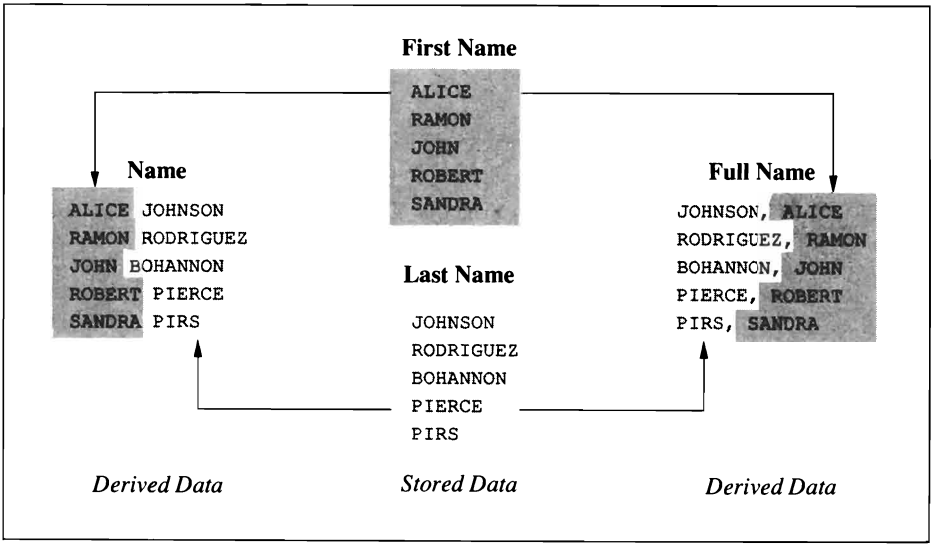


Figure 8-2. The C Code Concatenates Stored Data.
The C correlative can concatenate last names and first names in any order.

Example 8-2 shows what the data derived by this correlative would look like if it were output in a report.

```
>LIST CUSTOMERS LAST-NAME FIRST-NAME FULL-NAME
   NAME HDR-SUPP ID-SUPP
```

Last Name.	First Name	Full Name.....	Name.....
JENKINS	HAROLD	JENKINS, HAROLD	HAROLD JENKINS
BOHANNON	JOHN	BOHANNON, JOHN	JOHN BOHANNON
BROWN	JAMES	BROWN, JAMES	JAMES BROWN
BUCKLER	JULIE	BUCKLER, JULIE	JULIE BUCKLER
LEARY	BILL	LEARY, BILL	BILL LEARY
MASON	JULIA	MASON, JULIA	JULIA MASON
ORLANDO	AMY	ORLANDO, AMY	AMY ORLANDO
PIRS	SANDRA	PIRS, SANDRA	SANDRA PIRS
ASH	MARY	ASH, MARY	MARY ASH
EDWARDS	ANTHONY	EDWARDS, ANTHONY	ANTHONY EDWARDS
PEERCE	JAN	PEERCE, JAN	JAN PEERCE
PIERCE	RICK	PIERCE, RICK	RICK PIERCE
JOHNSON	ANNE	JOHNSON, ANNE	ANNE JOHNSON
JOHNSON	HENRY	JOHNSON, HENRY	HENRY JOHNSON
HIGGINS	HENRY	HIGGINS, HENRY	HENRY HIGGINS
EDGECOMB	DAVID	EDGECOMB, DAVID	DAVID EDGECOMB
MEADE	ANDREW	MEADE, ANDREW	ANDREW MEADE
LAMPSON	BOB	LAMPSON, BOB	BOB LAMPSON

Example 8-2.
Data in FULL-NAME and NAME is derived from the attributes LAST-NAME and FIRST-NAME.

The stored data is listed first (LAST-NAME, FIRST-NAME), then the concatenated full name in the same order (FULL-NAME) and in reverse order (NAME).

The correlative that does this is fully described in the section, "Concatenating Data (The C Code)."

Yet other correlatives can be used to extract specified portions of data from an attribute. To reverse the preceding example, if people's full names are stored in one attribute in the order *first-name last-name*, a correlative can extract just the last names. Or both the last and the first names can be extracted separately, then listed in the order *last-name first-name*.

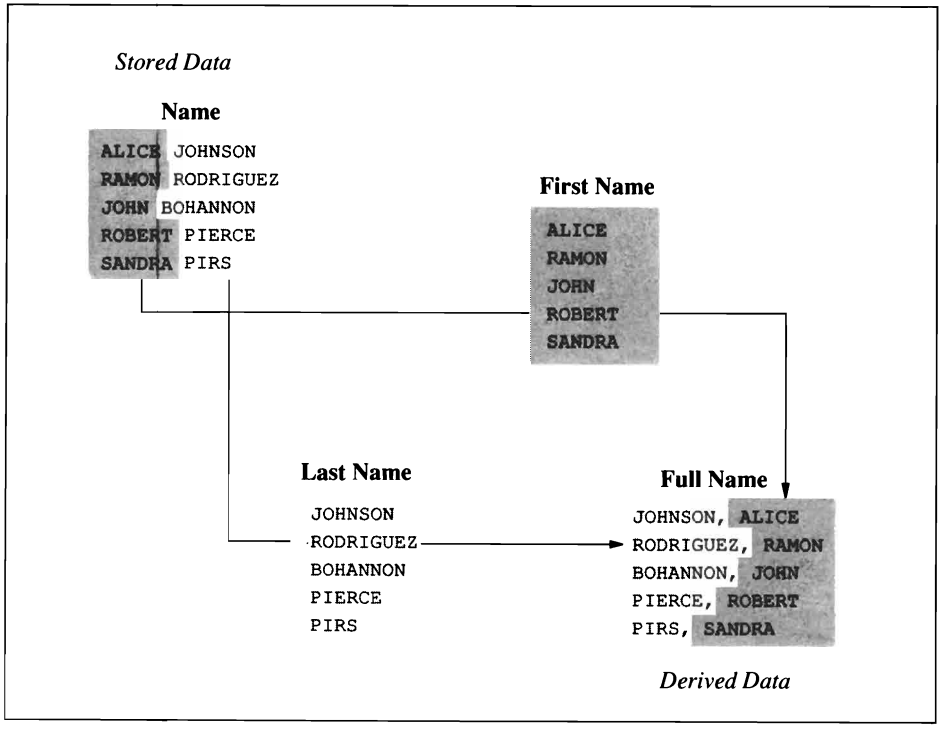


Figure 8-3. The G Correlative Extracts Data.
The G code can extract last names from full names. Or both first and last names can be extracted, then a C code can rearrange them in another order.

Example 8-3 shows what the data derived by this correlative would look like if it were output in a report.

```
>SORT CUSTOMERS BY LAST-NAME NAME LAST-NAME
  FULL-NAME HDR-SUPP ID-SUPP

Name.............. Last Name. Full Name.........

MARY ASH            ASH        ASH, MARY
JOHN BOHANNON       BOHANNON   BOHANNON, JOHN
JAMES BROWN         BROWN      BROWN, JAMES
JULIE BUCKLER       BUCKLER    BUCKLER, JULIE
DAVID EDGECOMB      EDGECOMB   EDGECOMB, DAVID
ANTHONY EDWARDS     EDWARDS    EDWARDS, ANTHONY
HENRY HIGGINS       HIGGINS    HIGGINS, HENRY
HAROLD JENKINS      JENKINS    JENKINS, HAROLD
ANNE JOHNSON        JOHNSON    JOHNSON, ANNE
HENRY JOHNSON       JOHNSON    JOHNSON, HENRY
BOB LAMPSON         LAMPSON    LAMPSON, BOB
BILL LEARY          LEARY      LEARY, BILL
```

Example 8-3.
Data in LAST-NAME and FULL-NAME is derived from the attribute NAME.

The correlatives that perform these operations extract the desired data from the attribute in which it is stored. They are described in the section, "Extracting Data."

Correlatives can also be used to verify or test data, ensuring that only data specified by the correlative be selected, processed, or listed in a report. For example, there are correlatives that test whether data is of a certain length, or whether data falls within a specified range, or whether data matches a specified pattern. These correlatives are described in the section, "Verifying Data."

Finally, there is a very powerful code, T*file*, that makes it possible to access data stored in other files and use it for processing and output. In effect, related data stored in different files can be linked together, allowing you to use data stored in multiple files to produce complex ACCESS reports. The code that does this is described in the section, "Translating Data from Another File (T*file*)."

Conversions

Conversions change the format of either stored or derived data, converting it to an *external format* suitable for listing in ACCESS reports. For instance, the derived amount of an employee's pay as calculated by the arithmetic correlative described in the preceding section would be unformatted data:

```
9673279
```

A conversion code can be applied to this number so that when it is output in a report, it is formatted as a dollar amount rounded off to two decimal places, complete with a dollar sign:

```
$967.33
```

Other conversion codes allow you to output times and dates in a number of different formats. For example, date conversions allow you to specify such formats as 19 JUL 1991, 07/19/91, 07-19-1991; time conversions let you specify formats such as 13:30, 01:30PM, or 01:30:00.

Still other conversion codes can be used to change lowercase characters to uppercase and vice versa, to translate decimal numbers and ASCII character strings to their hexadecimal equivalents (and vice versa), and to extract certain categories of characters for output, such as all alphabetic or all numeric characters, from stored data that includes both categories.

Conversions are used not only to convert stored and derived data into external format; they are also used to convert literals in an ACCESS query from external format into internal, or stored, format. For instance, if you entered a query such as the following:

```
>SORT ACCOUNTS WITH DATE <= "11/14/89"
```

a conversion code in the Attribute Definition item DATE would convert 11/14/89 into the system's internal date format (in this case, 7989).

Conversion codes accomplish several purposes. Since conversions format data only when it is output, data can be stored without accompanying formatting characters such as dollar signs, commas, and decimal points, thus taking up less disk space. Operations such as sorting and comparing take less time when they are performed on unformatted data. The external format of large amounts of data can be changed easily by editing just one conversion code in the appropriate Attribute Definition item; changes need not be made to each piece of stored data.

Conversion codes are fully described in the section, "Formatting Data."

How Correlatives and Conversions Are Applied

Codes placed in Attribute 7 of an Attribute Definition item are applied as conversions, and codes placed in Attribute 8 are applied as correlatives.

The main difference between correlatives and conversions is the way in which ACCESS applies them. The general rule is: correlatives are applied before conversions. To be more specific, correlatives are applied to data in an item immediately after it is read from a file, *before* it is selected or sorted (or otherwise processed). Conversions, on the other hand, are applied to literals in the command line and to data *after* it is processed, just before it is output in a report. Figure 8-4 illustrates this sequence of operations.

When an ACCESS statement is executed, raw data is taken from the database and any correlative code is applied to it. As has already been said, the information that results is derived data. Derived data is used:

- in selecting.
- in sorting.
- in totalling.
- in producing control breaks.
- in printing (except on break lines).

Once any of the above processing is done, a conversion code can be applied, producing an external format for the data. The conversion reformats the data into a form suitable for the ACCESS report, which can either be displayed on the screen or run off on the printer.

Because correlative and conversion codes are placed in the file dictionary associated with the data they affect, the stored data is not changed by any of the preceding operations. In other words, correlative and conversion codes are newly applied to the data each time an ACCESS report is generated.

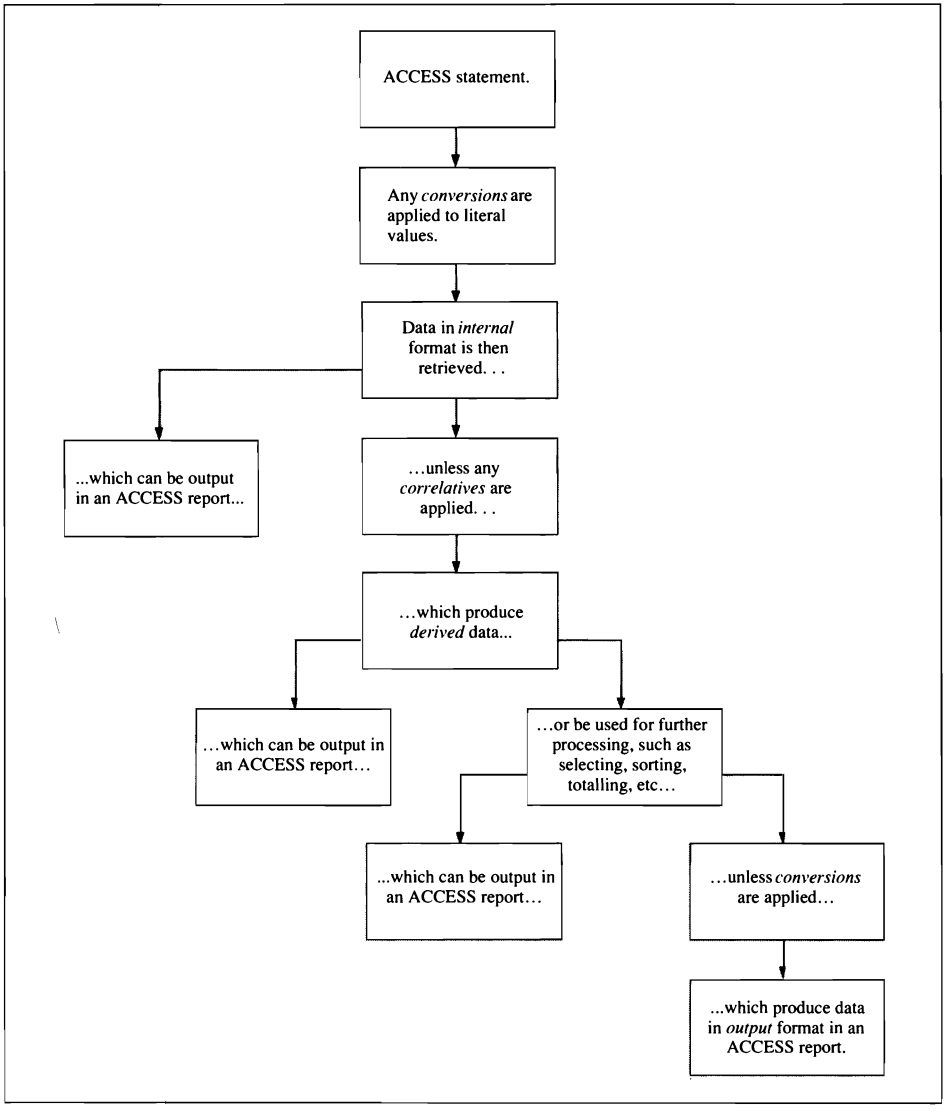


Figure 8-4. When Correlatives and Conversions are Applied.
Correlatives and conversions are applied to data in stored format at the time an ACCESS report is generated.

Performing Arithmetic Operations

The two codes that perform arithmetic or string processing are the A and F codes. The A code is the easier of the two to use.

Manipulating Numeric Data and Strings (A Code)

The A code is used to manipulate literals, system variables, and numeric or string data located in other attributes. It uses the following syntax:

A [**;**]*expression*

An *expression* can be one or more arithmetic or relational operators which operate on any of the following operands: *attributes* specified either by attribute number (AMC) or by attribute name; *literals* in single or double quotes; special *system variables*; or *functions*. Use parentheses to indicate the precedence of operations.

A maximum of 20 levels of nesting parentheses are permitted with the A code. If parentheses are not used, the order of precedence of operations is as follows:

1. Multiplication and division.
2. Addition and subtraction.
3. Relational operators.

If two operators have the same precedence, they are applied from left to right.

The following example demonstrates how to apply a simple A correlative. A book store's ORDERS file contains the following A correlative to calculate the total amount of sale for any title ordered:

```
A ; 2 * N ( PRICE )
```

This correlative multiplies the data stored in Attribute 2 (the number of copies sold) by the data from the attribute PRICE.

Figure 8-5 identifies each element in this correlative code.

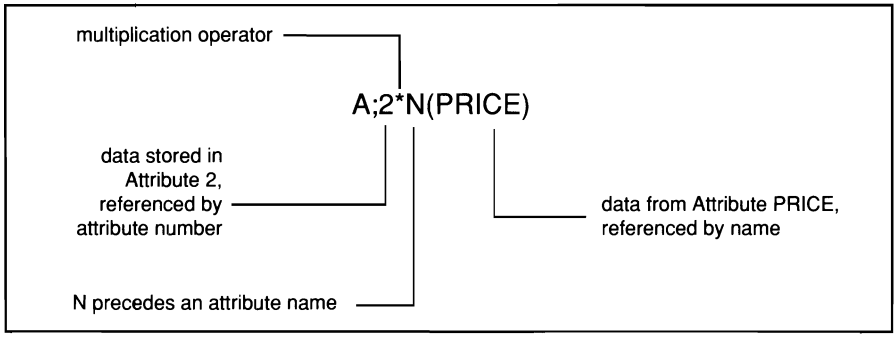


Figure 8-5. The A Code.
This A correlative calculates the amount of sale for a book by multiplying data stored in Attribute 2 (quantity sold) by data from the attribute PRICE.

In the example, Attribute 2, QTY, is referenced by its attribute number (2), but the attribute PRICE is referenced by its attribute name. When referencing an attribute by its number, or AMC, just the number itself is needed. When referencing an attribute by its name, however, the name must be enclosed in parentheses and preceded by the letter "N". ACCESS handles the data stored in these two attributes differently because of how they are referenced.

If you specify an attribute by its attribute number, the values in that attribute position are used just as they are stored. If, however, you specify the attribute by its attribute name (the item ID of the Attribute Definition item in the file dictionary), any correlatives are applied before processing.

In the example, Attribute 2 is referenced by its attribute number because the data is stored in attribute position 2 in the file. The attribute PRICE, however, is referenced not by its number but by its name, because its data is not stored in the ORDERS file at all but is derived from the BOOKS file. Price data is obtained from the BOOKS file using a T*file* correlative. The T*file* correlative is located in the ORDERS dictionary, in the Attribute Definition item for PRICE. (T*file* correlatives are described in detail later in this chapter.)

When attribute names are specified, they must be enclosed in parentheses and preceded by the letter N, as shown in the example.

The Attribute Definition item for the attribute AMOUNT contains the following data:

```
     AMOUNT
001  S
002  0
003
004
005
006
007  MR2$
008  A ; 2*N(PRICE)
009  R
010  10
```

The attribute number (line 2) shown for AMOUNT is 0. This is because the data is derived, not stored in the file in a specified attribute position. Since the Attribute Definition item for AMOUNT does not define a real attribute (that is, one that contains data stored in the file), it is sometimes referred to as a *dummy* attribute. Any number can be used as the attribute number of a dummy attribute; using high numbers such as 99 or 999 is a frequently seen convention. Do not, however, use 9999, as this is reserved. (see the section, "Advanced Topics," later in this chapter)

Constants included in an A correlative must be enclosed in single or double quotes. For example, the following A correlative multiplies the data in the attribute PRICE by the number 2, not by data contained in Attribute 2:

```
A ; "2" * N ( PRICE )
```

The A correlative in the next example calculates the total sale for an order (consisting of multiple copies of more than one book):

```
A ; S ( N ( AMOUNT ) )
```

The S function shown in the preceding correlative sums all of the data in the specified multivalued attribute (in this case, AMOUNT).

The arithmetic operators that can be used with the A correlative are shown in Table 8-1.

Table 8-1. Arithmetic Operators.

Operator	Meaning
+	Addition.
–	Subtraction.
*	Multiplication.
/	Division (returns an integer).
:	Concatenation

The relational operators are shown in Table 8-2. You can also use the boolean operators AND and OR. You cannot use & (and) or ! (or).

Table 8-2. Relational Operators.

Operator	Meaning
=	Equal to.
#	Not equal to.
>	Greater than.
<	Less than.
>=	Greater than or equal to.
<=	Less than or equal to.

System functions that can be used with the A correlative are shown in Table 8-3.

Table 8-3. System Functions.

Function	Operation
R (*operand1*,*operand2*)	returns the remainder after division.
S (*expression*)	sums multivalues.
attribute ['*start* ', '*length* ']	extracts a substring.
IF...THEN...ELSE	conditional expression.

The R function returns the remainder after dividing the first operand by the second operand. *operand* can be any valid expression.

The S function sums all multivalues in *expression.*

Brackets can be used to extract a substring from an attribute. *start* and *length* must be enclosed in quotes. *attribute* can be identified either by attribute number or by attribute name. Attribute numbers, literals in single or double quotes, or expressions can be specified within the brackets. Brackets are part of the syntax here and must be typed. *start* is the starting position of the first character to be extracted, *length* is the number of characters to be extracted.

For example, the following A correlative returns the fourth and fifth digits of the attribute named SS-NUM:

```
A ; N(SS-NUM) [ '4' , '2' ]
```

A conditional statement can be included in an A correlative. The syntax is:

IF *expression1* **THEN** *expression2* **ELSE** *expression3*

This function evaluates *expression1* and, if true, then evaluates *expression2*, or if false, evaluates *expression3*.

The following special system functions can be specified in an A (or F) correlative:

NB	NI	NS	9998
ND	NV	LPV	9999

These special counter operands can be used to count break levels, detail lines, items, etc. The LPV operator can be used to load the value obtained from a previous correlative operation into another correlative for further processing. Since most of the codes that use these special functions are used as conversions in line 7 rather than as correlatives in line 8, they are discussed in the section, "Advanced Topics," later in this chapter.

Table 8-4 shows more examples of how the A correlative is used.

Table 8-4. Using the A-Code.

A Code	What it does
A ; 1+2	Adds Attributes 1 and 2.
A ; "10"*3	Multiplies each value of Attribute 3 by 10.
A ; S(4+"25")	Adds 25 to each value of Attribute 4, then sums the multivalues.

A ; N(INV-AMT)-N(BAL-DUE)	Subtracts the value of the attribute BAL-DUE from the value of the attribute INV-AMT.
A ; N(SS-NUM)['4','2']	Returns the fourth and fifth digits of the attribute SS-NUM.

Using the Stack (F Code)

The F code performs mathematical and string processing operations on constants or on data stored in specified attributes. F correlatives are made up of operands and operators in reverse Polish notation (Lukasiewicz high Polish) separated by semicolons. Unlike the A code, the F code cannot use dictionary names to reference attributes.

The F code performs operations on the last one or two entries pushed onto a stack. The result of the operation is placed at the top of the stack.

Figures 8-6, 8-7, and 8-8 illustrate the way the stack processes elements pushed onto it by the following F correlative:

F ; C3 ; C4 ; C5 ; * ; + ; C6 ; –

This simple example uses only constants and operators; it is intended only to show how stack operations are performed. Each figure shows what happens when one of the operators is pushed onto the stack.

In Figure 8-6, three elements are placed on the stack. The multiplication operator acts on the top two elements, placing the result of 20 on top of the stack. Notice that the result replaces the two operands.

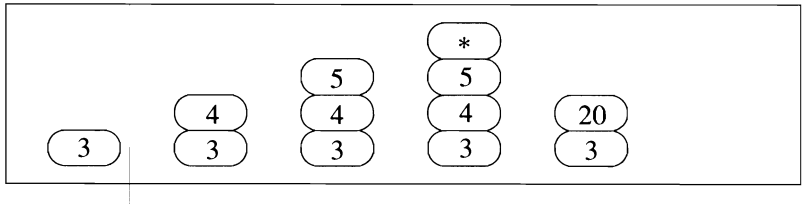


Figure 8-6. The Multiplication Operator.

In Figure 8-7, the addition operator acts on the two elements in the stack, 3 and 20. The result, 23, is pushed onto the stack.

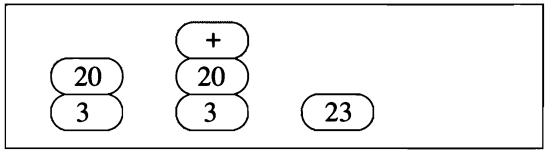


Figure 8-7. The Addition Operator.

In Figure 8-8, the subtraction operator causes the top element in the stack, 6, to be subtracted from the second element, 23. At the end of these operations, the result, 17, is the only element remaining in the stack.

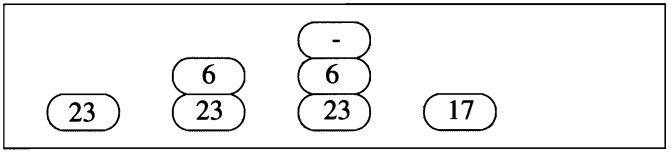


Figure 8-8. The Subtraction Operator.

The F code uses the following syntax:

F [**;**] *element* [**;** *element* ;...]
FS [**;**] *element* [**;** *element* ;...]

Notice that there are two versions of the F code. The earlier version, F, differs from the FS ("standard") version mainly in the way it handles relational operators. See the section, "Relational Operators," later in this chapter.

ACCESS parses the expression from left to right, putting each element on the stack as an operand until it encounters an operator. The operation is then performed.

An *element* can be any of the expressions described in the following sections. Many of these are the same as those defined for the A correlative in the preceding section.

Referencing Attributes and Literal Strings

Any of the following can be operands:

attribute# [**R** [**R**]]
"*literal* **"**
C*literal*

Attributes can be referenced only by attribute number, not by attribute name. This is different from the A code, where you can use either one. For example, if the number of items sold are stored in Attribute 5 and unit prices are stored in Attribute 2, the following F correlative can be used to derive the total amount of sale:

F ; 2 ; 5 ; *

It is not possible to enter this as "F ; N(QTY) ; N(PRICE) ; * ". You can, however, use the following A correlative to perform the same operation:

A ; N(QTY) * N(PRICE)

The optional R specifies that the first value of the attribute be used repeatedly with multivalued attributes. A second R specifies that the first subvalue be used repeatedly with other subvalues.

Literal strings can be specified in two ways, either by enclosing them in single or double quotes, or by preceding them with the letter "C". For example, either the element " "4" " or the element "C4" pushes the number 4 onto the stack.

Arithmetic Operators

Any of the following arithmetic operators can be used:

+	addition.
–	subtraction.
*[*n*]	multiplication.
/	division.

Arithmetic operators affect the top two elements in the stack. The operands are popped and the result is pushed on to stack as the new first element. – (minus) subtracts the top element from the second element. / (slash) divides

the second element by the top element. The *n* after the multiplication operator divides the result by 10 raised to the *n*th power.

Table 8-5 compares the use of arithmetic operators for the A and F codes.

Table 8-5. Arithmetic Operators for A and F Codes.

Operation	A Code		F Code	
20 X 4	A;"20"*"4"	F;C20;C4;*	or	F;"20";"4";*
20 – 4	A;"20"–"4"	F;C20;C4;–	or	F;"20";"4";–
20/4	A;"20"/"4"	F;C20;C4;/	or	F;"20";"4";/

Relational Operators

The following relational operators can be included in an F code:

=	equal to.
#	not equal to.
>	greater than.
<	less than.
]	greater than or equal to.
[	less than or equal to.

You can also use the boolean operators & (and) and ! (or). You cannot use the words AND or OR. Relational operators affect the top two elements in the stack. These operators return to the top of the stack a result of 1 if the condition is satisfied or 0 if the condition is not satisfied.

The following operators are handled differently depending on whether your system supports the F or the FS version of the F code:

<	less than.
>	greater than.
[	less than or equal to.
]	greater than or equal to.

The F version compares the *top* element of the stack to the *second* element. If the top element is less than (or greater than, etc.) the second element, the condition is TRUE and a 1 is pushed onto the stack. The FS version, on the other hand, compares the *second* element to the *top* element. In this case, if

the second element is less than (or greater than, etc.) the top element, the condition is TRUE and a 1 is pushed onto the stack.

Special Function Codes

The following special function codes can be included in an F code:

Table 8-6. Special Function Codes.

Code	Description
_	(underscore) Exchanges the first and second elements in the stack.
:	Concatenates the first element in the stack to the end of the second element.
P	Duplicates the first element by pushing it onto the stack.
R	Divides the first element in the stack by the second element, putting the remainder on top.
S	Sums multivalues of the first element of the stack.
[ ]	Extracts a substring by operating on the top three elements in the stack, where: Top *length* 2 *start* 3 *string* The third element on the stack is the string to be operated on. The second element is the character position at which the string starts. The top element is the length or the number of characters to extract. The result is placed on top of the stack.
(*conversion*)	applies a conversion on the first element in the stack.

The section, "Advanced Topics," at the end of this chapter lists the other system variables that can be used with the F code.

Applying Conversion Codes

A conversion code specified in parentheses will be applied to the top element in the stack. For example, the following F correlative divides data values

stored in Attribute 6 by the constant 2, then applies an MR conversion code to format the result:

```
F ; 6 ; C2 ; / ; ( MR2 )
```

The next example uses date conversions to return the number of years' difference between today's date and dates on file (in other words, each date on file is subtracted from today's date):

```
F ; D ; (DY) ; 3 ; (DY) ; –
```

The system variable "D" provides today's date in internal format; the date conversion "DY" is applied to it, pushing the current year onto the stack. Next, a date stored in Attribute 3 is pushed onto the stack and the DY conversion is applied to it. The subtraction operator is then applied, subtracting the referenced year (the top element) from the current year (the second element).

Deriving Data from Attributes

There are several codes that derive information from data stored in other attributes. These codes do the following:

- Concatenate data from two or more attributes (C).
- Extract delimited fields (G).
- Extract a specified number of characters (T).
- Substitute a string or the data in an attribute for another attribute (S).

Other codes can be used to test data and return only those elements that:

- Meet the specified length in characters (L).
- Match the specified pattern of alphabetic, numeric, and special characters (P).
- Fall within a specified range of values (R).

The following sections contain examples of these codes.

Concatenating Data (C Code)

The C code concatenates the data in two or more attributes. Literals can also be included in the concatenation.

For example, in the CUSTOMERS file, LAST-NAME is Attribute 2 and FIRST-NAME is Attribute 1. The Attribute Definition item FULL-NAME defines a new attribute that concatenates the last and first names (in that order). This is done by including the following C correlative in line 8 of the Attribute Definition item FULL-NAME:

```
C2 ; ', ' ; 1
```

"C2;', ';1" concatenates data contained in Attribute 2 with data contained in Attribute 1, separating the two elements with a comma followed by one space. If a customer's last name is Morris and first name is Steven, the correlative produces the following result:

```
MORRIS, STEVEN
```

On the other hand, the following C code:

```
C1  2
```

would produce the following result:

```
STEVEN  MORRIS
```

In the preceding example, a space is entered between the two expressions as a separator

The syntax of the C code is:

```
Cexpression1 ch  expression2 [ ch expression3... ]
```

expression can be any attribute number, any literal string enclosed in single or double quotes or backslashes, or an asterisk. An asterisk specifies the result generated by a previous conversion or correlative operation. *ch* can be any nonnumeric character with the exception of the semicolon (;), which specifies that *no* separation character is to be used. No spaces should be included between *expression* and *ch*.

In the next example, a C correlative builds a part number from the three pieces of data contained in Attributes 3, 2, and 4 of an inventory file. Each part specification begins with the literal string "AZ" and uses hyphens as separation characters.

Here is the Attribute Definition item for PART#:

```
0001 A
0002 0
0003 Part No.
0004
0005
0006
0007
0008 C'AZ';3-2-4
0009
0010 13
```

The literal string "AZ" is enclosed in quotes. The semicolon specifies that no space (or other separator) is to be used between the literal expression and the data of Attribute 3. Values from Attribute 3, Attribute 2, and Attribute 4 are then specified, to be separated from each other by hyphens.

Here are lines 2, 3, and 4 of the item SERVO in the PARTS file:

```
002 35
003 250
004 1002
```

When the C correlative is applied to the data in this item and output in a report, it looks like this:

```
PARTS..... Part No......

SERVO      AZ250-35-1002

END OF LIST

>
```

Example 8-4.
A C correlative creates the part number from the literal string 'AZ' and three attributes.

Any Attribute Definition item containing a C correlative should specify an attribute number of zero in Attribute 2. If any other attribute number is used and that attribute contains a null value, the concatenation will not be performed.

The following table shows more examples of how the C correlative can be used. The item referenced contains the following data:

```
001 KANE
002 MIKE
003 514
004 MANAGER
005 COMMUNICATIONS
006 N/L
007 N/B
008 M3270
```

A ◊ indicates a space character.

Code	Result
C1 , 2	KANE,MIKE
C1 ; ' ,◊ ' ; 2	KANE, MIKE
C " EXT◊ " ; 3	EXT 514
C1 ; \ , " \ ; 4 ; ' OF ' ; 5 "	KANE, "MANAGER OF COMMUNICATIONS"

Extracting Data

Two codes that extract data from an attribute are the T code (text extraction) and the G code (group extraction). The T code extracts a specified number of characters. The G code extracts groups of characters that are separated by delimiters.

The T Code

The following example illustrates the use of the T code. The BOOKS file contains the full title of each available book in the attribute TITLE. Titles can be up to 20 characters long. Since an abbreviated form of each book title needs to be included in certain ACCESS reports, the T code is used to extract the first eight characters from attribute TITLE.

The syntax of the T code is as follows:

T [*start* ,] *#chars*

where *start* is the starting position where the extraction is to begin, and where *#chars* is the number of characters to extract. If *start* is not specified, justification determines whether the count is from left to right (L justification) or from right to left (R justification). If *start* is specified, characters are extracted from left to right.

Thus, the Attribute Definition item for SHORT-TITLE in the BOOKS file contains the following data:

```
     SHORT-TITLE
001  S
002  2
003
004
005
006
007
008  T8
009  L
010  8
```

SHORT-TITLE is defined as a synonym for Attribute 2, TITLE, from which the T correlative extracts the first eight characters. For example, if the full titles were:

```
DATABASE MANAGEMENT SYSTEMS
OPERATING SYSTEM CONCEPTS
WORD PROCESSING
WRITING COMMERCIAL APPLICATIONS
```

the short titles would be:

```
DATABASE
OPERATIN
WORD PRO
WRITING
```

In the next example, Attribute 3 contains a person's complete phone number, including a 3-digit extension, stored in the format:

```
718-538-9000X512
```

The following Attribute Synonym Definition item could be used to extract just the extension number "512":

```
     EXTENSION
001 S
002 3
003
004
005
006
007
008 T3
009 R
010 3
```

Extraction occurs from right to left because no starting position is specified and the justification is "R". Similarly, to extract the area code "718", use the same T correlative of "T3" and specify left justification in line 9.

Yet another Attribute Synonym Definition item, named MAIN-PHONE, could be used to extract the main number (538-9000) without the area code or extension:

```
     MAIN-PHONE
001 S
002 2
003
004
005
006
007
008 T5 , 8
009 R
010 8
```

In this case, the extraction begins at the fifth character and takes the next eight characters, moving left to right. Because the starting position of the extraction is specified, the extraction occurs from left to right, regardless of the justification specified in line 9.

Table 8-7 shows more examples of how the T correlative can be used. A ◊ represents a space.

Table 8-7. Using The T Code

Stored Value	Justification	T Correlative	Output
ABCDEF	L	T3	ABC
ABCDEF	R	T3	DEF

HELLO◊OUT◊THERE	L	T3,5	LLO O
HELLO◊OUT◊THERE	R	T3,5	LLO O
123SMITH◊◊CR	L	T4,7	SMITH
848JOHNSONDB	L	T4,7	JOHNSON
123SMITH◊◊CR	L	T3	123
848JOHNSONDB	L	T3	848
123SMITH◊◊CR	R	T2	CR
848JOHNSONDB	R	T2	DB

The G Code

The G code extracts groups of characters that are separated by delimiters. Since words are separated by spaces, the G code can be used to extract words from a sentence.

The syntax for the G code is as follows:

G [*skip*] *delimiter #fields*

where *skip* is the number of fields to skip, *delimiter* is the character used to separate fields, and *#fields* is the number of contiguous fields to extract. *delimiter* can be any single nonnumeric character except a system delimiter (segment mark, attribute mark, value mark, subvalue mark, or start buffer mark).

When you create an Attribute Definition item with a G code that is to extract data from an attribute that already exists, the attribute number (line 2) of the new Attribute Definition item should be the same as that of the existing attribute.

Continuing the example in the preceding section, you could use a G code instead of a T code to derive the short title from the BOOKS file. The G code "G 1" extracts the first word from the attribute TITLE. The space between "G" and "1" specifies the space character as the delimiter. "1" specifies that only the first field is to be extracted.

"G 1" produces the following data:

```
DATABASE
OPERATING
```

```
WORD
WRITING
```

Another example: Attribute 2 in a personnel database contains a department ID and a job ID delimited by a slash (/). The data is stored in the following format:

```
30/110
```

The G correlative in the following Attribute Definition item extracts just the department ID:

```
     DEPT.ID
001 S
002 2
003 Dept ID
004
005
006
007
008 G/1
009 L
010 3
```

"G/1" extracts only one field, the first one; the delimiter is defined as a slash (/). Since the number of fields to skip before extraction is not specified before the delimiter, a default of zero is assumed. "G1/1" would specify that the first field should be skipped and only one field, the second one, should be extracted.

In an attribute with several fields separated by asterisks, "G1*3" would specify that the first field should be skipped and the next three fields (2, 3, and 4) should be extracted.

Table 8-8 shows some more examples of how the G correlative works. The stored data for all these examples is "10/20/30/40".

Table 8-8. Using The G Code

If the G Correlative is...	Then Data Extracted is...
G/1	10
G/2	10/20
G1/1	20
G1/2	20/30
G2/1	30
G4/1	<null>

Substituting Data (S Code)

The S code substitutes for the value of an attribute either the *data* in another attribute or a specified *character string*. The attribute whose data is to be replaced is the one referenced by the attribute number in line 2 of the Attribute Definition item containing the S code.

If a value in the referenced attribute is null or zero, the first element specified in the S code is substituted. If a value in the referenced attribute is *not* null or zero, then the second element specified by the S code is substituted.

The syntax of the S code is as follows:

S ; *element1* ; *element2*

element can be either an attribute number or a literal string. Literal strings must be enclosed in single quotes. If the value in the referenced attribute is *not* null or zero, then *element1* is substituted for the value. If the value of the referenced attribute *is* null or zero, then *element2* is substituted for it. To retain the original value of the referenced attribute (the attribute specified on line 2), use an asterisk (*) in place of *element1* or *element2*.

For example, the Attribute Definition item for Attribute 6, PAYMENT, in the OVERDUE.ORDERS file contains the following correlative:

```
S ; 3 ; 'NONE'
```

This correlative specifies that if the value in Attribute 6 is null or zero, the string "NONE" is substituted. If the value in Attribute 6 is not zero, then the data in Attribute 3, CHECK-AMOUNT, is substituted.

The S correlative thus makes it possible to bypass the original (out-of-date) data in the PAYMENT attribute.

Use the S code along with the F correlative to test for zero and then take different actions according to what kind of data it encounters. For example, F;1(S;*; 'NORMAL VALUE') specifies that if a value in Attribute 1 is zero, the string "NORMAL VALUE" will be used; otherwise the original contents of Attribute 1 is to be used.

Testing Data

The L (length), P (pattern), and R (range) codes test data before including it in an ACCESS report. This section summarizes the operations performed by these three codes.

The L Code

The L code verifies data based on its length and produces no output if the test fails. For example, assume that the (unformatted) price of a book in the BOOKS file must be at least one character and no greater than three characters in length. The following L correlative verifies the data in the attribute PRICE before including it in a report:

```
L1 , 3
```

One (1) specifies that the value must contain at least one character, 3 specifies that the value should be no longer than three characters.

The syntax of the L code is as follows:

L [*n* [,*m*]]

where *n* specifies a maximum length of *n* characters and *n,m* specifies a range of from *n* to *m* characters. Data either exceeding *n* characters or falling outside the range of *n–m* produces a null value. If *n* is zero, or if L is used without any parameters, the length of the value is returned.

The R Code

The R code specifies a single numeric range or multiple numeric ranges within which the data must fall. The syntax of the R code is as follows:

R *n* , *m* [; *n* , *m* ;...]

where *n* is the lower bound and *m* is the upper bound.

If multiple ranges are specified, the data which falls within any of these ranges is included in the ACCESS report. Ranges should be specified in

ascending order. For example, the correlative R3,5;9,14;16,30 includes data that falls within the ranges 3-5, 9-14, or 16-30.

The P Code

The P code compares data to a specified pattern of numeric, alphabetic, alphanumeric, or literal characters. Patterns must be enclosed in parentheses and can be specified as shown in Table 8-9.

For example, the following P correlative ensures that only Social Security numbers will be included in ACCESS reports:

```
P ( 3N-2N-4N )
```

The preceding P code tests for strings of exactly three numbers, then a hyphen, then exactly two numbers, then a hyphen, then exactly four numbers.

The syntax of the P code is as follows:

P (*pattern* **) [; (** *pattern* **)...]**

pattern can be specified using any of the specifications in Table 8-9.

Table 8-9. Pattern Matching Specifications.

Pattern	Description
*n*A	An integer followed by the letter A, which tests for *n* alphabetic characters.
*n*N	An integer followed by the letter N, which tests for *n* numeric characters.
*n*X	An integer followed by the letter X, which tests for *n* alphanumeric characters.
'*string* '	A literal string, which tests for that literal string. The string must be enclosed in single quotes. Single-character delimiters need not be treated as literal strings—that is, they need not be enclosed in quotation marks.

A semicolon can be used to test for more than one pattern. For example, if you wanted to test for telephone numbers of 7, 10, and 11 digits, you could use the following P correlative:

```
P ( 3N-4N ) ; ( 3N-3N-4N ) ; ( 1N-3N-3N-4N )
```

Table 8-10 shows some examples of the P code.

Table 8-10. Using the P Code.

If *pattern* is...	then output is...
'617'-3N-4N	Any phone number with a 617 area code.
2N/2N/2N	Any field that resembles a date in external format.
2N:2N:2N	Any field that resembles the time in external format.
5A5X	A string comprised of five alphabetic characters followed by five alphanumeric characters. (For example, a Customer-ID might be made up of the first letter of the customer's first name, the first four letters of the last name, and the first five characters of the street address.)
'10'3N	Any five-digit order-ID that begins with "10".

Translating Data from Other Files

The T*file* code makes it possible to access data in other files. This powerful code eliminates the need for duplicating data in related files.

The T*file* code in the *source* file references data in another file, the *target* file. The source file must include keys to the target file—that is, data which is identical to the item IDs of the target file. These keys can be contained in one of the attributes of the source file, or they can be built from data stored in two or more attributes. These keys allow the source file to access any of the data in the target file. (See Figure 8-9.)

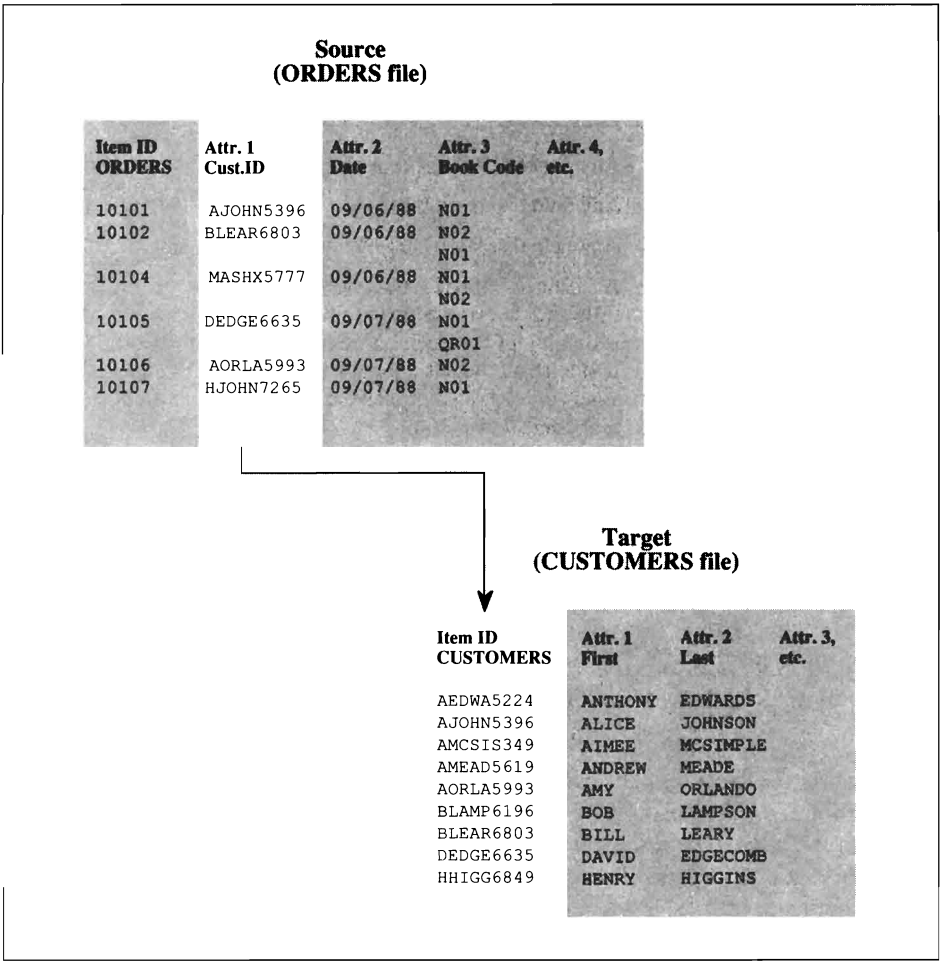


Figure 8-9. The Source File and the Target File.
Attribute 1 in the source file contains the keys to the items in the target file.

The T*file* code is placed in its own Attribute Definition item in the source file. This Attribute Definition item is created as a synonym for the Attribute Definition item (also in the source file) that defines the attribute mentioned in the preceding paragraph—the one that contains the item IDs of the target file. This is done by assigning an A or an S (synonym) code in Attribute 1 and by

assigning an attribute number which is the same as the number of the attribute containing the target file's item IDs.

The T*file* code itself specifies (1) the *name* of the target file, (2) the *number* of the attribute in the target file from which data is to be translated, and (3) what *action* is to be taken if there is no data in that attribute.

Here is an example of how T*file* correlatives can be used. A book store stores book orders in an ORDERS file and stores its book inventory in a BOOKS file. Each order in ORDERS includes book titles and prices for each book order, but titles and prices are not stored in ORDERS. Instead, using T*file* correlatives, the orders file "looks up" the titles and prices in BOOKS and uses them in ORDERS file reports. In this arrangement, ORDERS is the source file and BOOKS is the target file.

Example 8-5 shows some of the data stored in the ORDERS file.

```
PAGE   1                                          11:15:13  01 NOV 1989

ORDERS Qty Book Code

 10101   2 N01
 10102   3 N02
         1 N01
 10103   2 N01
 10104   3 N01
         5 N02
         1 QR01
        34 QR02
 10105   1 N01
         4 QR01
         7 QR02
 10106   3 N02
 10107   9 N01
        34 N02
 10108  10 N02
         2 QR01
 10109   7 QR02
 10110   3 N02
        45 QR01
         1 QR02
```

Example 8-5.
Data in attribute BOOKCODE references the item IDs of the BOOKS file.

The preceding report was generated by the following query:

```
>SORT ORDERS QTY BOOKCODE
```

The item IDs in BOOKS are unique codes (such as standard ISBN numbers) that identify each book. Attribute 1, BOOKCODE, of the ORDERS file contains the BOOKS item IDs, which identify by code number each book ordered. This attribute contains the keys that are used to access data in the BOOKS file.

Example 8-6 displays some sample output from the BOOKS file.

```
>SORT BOOKS AUTHOR TITLE PRICE
```

```
PAGE   1                                          11:15:28  01 NOV 1989

BOOKS.. Author.... Title.............................. Price...

N01     STRANG     DATABASE MANAGEMENT SYSTEMS            $9.95
N02     JACKSON    OPERATING SYSTEM CONCEPTS             $18.75
QR01    O'REILLY   WRITING COMMERCIAL APPLICATIONS       $24.50
QR02    BALLARD    WORD PROCESSING                       $22.98

4 ITEMS LISTED.

>
```

Example 8-6.
Some of the contents of the BOOKS file.

Using T*file* correlatives, you can link the data contained in the BOOKS file to the data contained in the ORDERS file, thus avoiding unnecessary duplication of data. The ORDERS file requires only one set of data to access the BOOKS file, namely the unique book code numbers which are the item IDs of BOOKS. These numbers are stored as Attribute 1, BOOKCODE, in the ORDERS file.

The Attribute Definition item for TITLE in the ORDERS file contains the following data:

```
   TITLE
001 S
002 1
003 Title
004
005
006
007
008 TBOOKS ; C ; ; 2
009 T
010 35
```

This Attribute Definition item is a synonym for the Attribute Definition item BOOKCODE in the source file ORDERS. It uses a definition code of S and an attribute number of 1, the same number used for the attribute BOOKCODE. The T*file* correlative accesses data stored in Attribute 2, TITLE, in the target file BOOKS.

Figure 8-10 identifies each of the elements in the T*file* correlative.

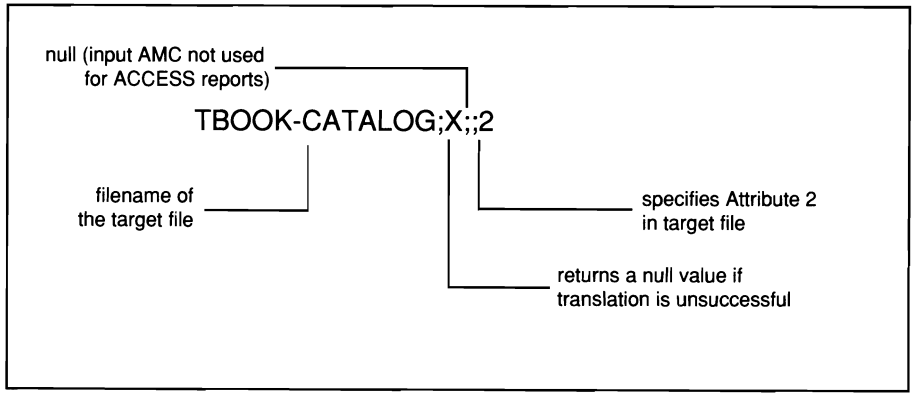


Figure 8-10. The T*file* Correlative.
This Tfile *correlative accesses the data in Attribute 2 of the target file.*

When a T*file* correlative is to be used by ACCESS, its syntax is:

T*filename* ; *code* ; ; *attribute* [; *break*]

where *filename* is the name of the target file, *code* specifies the action to be taken if no data is found, *attribute* is the number of the attribute in the target file that contains the data to be translated, and *break* is the optional number of the attribute in the target file to be translated for BREAK-ON and TOTAL lines. Two semicolons must separate *code* from *attribute*. These semicolons actually specify a null value for an input attribute. Input attributes are not used by ACCESS.

The T*file* code must include an *action code*. Action codes specify what action is to be taken if there is no data in the target file or if the item specified by the key does not exist in the target file.

The action code "X" shown in the preceding example specifies that if no value exists in Attribute 2 in the BOOKS file, or if the item specified by the item ID does not exist in the target file, a null value is displayed in the title column of the ACCESS report.

Table 8-11 shows the complete list of T*file* action codes with a description of what happens if the file translation can't take place.

Table 8-11. T*file* Action Codes.

Code	Description
C	Returns the item ID.
I	Verifies input only. Functions like V for input and like C for output.
O	Verifies output only. Functions like C for input and like V for output.
V	Returns the following error message: [708] *item-ID* cannot be converted and suppresses the entire print line for that item.
X	Returns a null value.

To continue the example, the ORDERS file uses another T*file* correlative to translate book prices from the BOOKS file. The correlative that retrieves book prices is just like the one used to retrieve titles, except that instead of the attribute TITLE, Attribute 3, PRICE, in the BOOKS file is referenced. The Attribute Definition item for PRICE in the ORDERS file dictionary contains the following data:

```
     PRICE
001 S
002 1
003 Price
004
005
006
007 MR2$
008 TBOOKS ; X; ; 3
009 R
010 8
```

This Attribute Definition item is created as another synonym for Attribute 1, BOOKCODE. This time, however, the T*file* correlative references Attribute 3, PRICE, in the BOOKS file. Also, since this attribute contains numeric values, it is right-justified.

Yet a third T*file* correlative is used to obtain the unit price from Attribute 3, PRICE, of BOOKS. This T*file* correlative is embedded in an F correlative that multiplies the unit price by the data in Attribute 2, QTY, of ORDERS to arrive at the total amount for each line item. The Attribute Definition item for LINE.AMT in the ORDERS file dictionary contains the following data:

```
     LINE.AMT
001 S
002 0
003 Item Amount
004
005
006
007 MR2$
008 F ; 1 ; (TBOOKS ; X ; ; 3) ; 2 ; *
009 R
010 10
```

With these three T*file* correlatives in place, it is now possible to replace the data in the attribute BOOKCODE with the titles and prices that are stored in the BOOKS file. Compare the orders report shown in Example 8-7 with the one shown in Example 8-5.

```
>SORT ORDERS QTY TITLE PRICE LINE.AMT

PAGE   1                                          11:15:44  01 NOV 1989

ORDERS Qty Title.............................. Price... Item Amount

 10101   2 DATABASE MANAGEMENT SYSTEMS            $9.95       $19.90
 10102   3 OPERATING SYSTEM CONCEPTS             $18.75       $56.25
         1 DATABASE MANAGEMENT SYSTEMS            $9.95        $9.95
 10103   2 DATABASE MANAGEMENT SYSTEMS            $9.95       $19.90
 10104   3 DATABASE MANAGEMENT SYSTEMS            $9.95       $29.85
         5 OPERATING SYSTEM CONCEPTS             $18.75       $93.75
         1 WRITING COMMERCIAL APPLICATIONS       $24.50       $24.50
        34 WORD PROCESSING                       $22.98      $781.32
 10105   1 DATABASE MANAGEMENT SYSTEMS            $9.95        $9.95
         4 WRITING COMMERCIAL APPLICATIONS       $24.50       $98.00
         7 WORD PROCESSING                       $22.98      $160.86
 10106   3 OPERATING SYSTEM CONCEPTS             $18.75       $56.25
 10107   9 DATABASE MANAGEMENT SYSTEMS            $9.95       $89.55
        34 OPERATING SYSTEM CONCEPTS             $18.75      $637.50
 10108  10 OPERATING SYSTEM CONCEPTS             $18.75      $187.50
         2 WRITING COMMERCIAL APPLICATIONS       $24.50       $49.00
 10109   7 WORD PROCESSING                       $22.98      $160.86
 10110   3 OPERATING SYSTEM CONCEPTS             $18.75       $56.25
        45 WRITING COMMERCIAL APPLICATIONS       $24.50     $1102.50
         1 WORD PROCESSING                       $22.98       $22.98
```

Example 8-7.
Data in TITLE, PRICE, and LINE.AMT is derived from the BOOKS file.

Multiple T*file* correlatives can be included in an Attribute Definition item. This allows data to be retrieved from a target file that is at more than one remove from the source file. For more information about multiple T*file* correlatives, see the section, "Using Multiple Correlatives," later in this chapter.

Formatting Data

Conversion codes are used to achieve consistency in the storage and output of data. They allow certain types of data to be stored in an internal format that can be easily and efficiently manipulated during operations such as selecting

and sorting. Conversions can then convert data into an external format suitable for ACCESS reports.

Conversion codes are also applied to literal values in ACCESS queries. The literal value is converted into its internal equivalent before the data is processed. Thus, even though a date is stored in internal format as "7555", you can access it with a query such as:

```
>LIST ORDERS WITH DATE = "9/6/88"
```

because the literal "9/6/88" will be converted into internal format by the conversion code.

Specifically, conversions can be used to format:

- Dates (D).
- Times (MT).
- Decimal numbers (ML and MR).

Conversions can also be used to perform the following operations:

- Lowercase/uppercase data conversion (MCL, MCU).
- Alphabetic/numeric character extraction (MCA, MCN).
- Decimal/hexadecimal number translation (MCDX and MCXD).
- Hexadecimal/ASCII character string translation (MX and MY).

The following sections describe how to use each of these conversion codes.

Formatting Dates (D Code)

The D code converts dates to stored format and upon output to one of many different external formats. Dates are stored internally as the number of days from the zero date, December 31, 1967 (for example, September 4, 1978, is stored as 3900).

The syntax of the D code is:

D [*year*] [{ *separator* | *subcode* }]

where *year* specifies the number of digits used to represent the year, *separator* is any nonnumeric character that separates the day, month, and

year, and *subcode* is a special code that specifies one of several date formats. Table 8-12 lists and summarizes the date subcodes.

Table 8-12. Date Subcodes.

Subcode	Description
D	lists only the number of the day.
I	lists date in internal format (reverse conversion). Date must be stored in one of the possible external formats.
J	lists only the Julian day of the year.
M	lists only the number of the month.
MA	lists only the name of the month.
Q	lists only the number of the quarter.
W	lists only the number of the day of the week (Sunday is 7).
WA	lists only the name of the day of the week.
Y	lists only the number of the year.

If either a nonnumeric separator or a subcode is specified, dates appear in the format 12/12/1967. If neither is specified, the format is 12 DEC 1967.

For example, the conversion code in line 7 of the Attribute Definition item DATE in the ORDERS dictionary is:

```
D2/
```

This conversion code specifies two digits to represent the year and the slash character (/) to separate the day, month, and year. For example, if the date in internal format is 8601 and the conversion "D2/" is applied, the external format is:

```
07/19/91
```

Table 8-13 shows some sample uses of the D code.

Table 8-13. Date Formats.

Conversion	Output
D	19 JUL 1991
D0	19 JUL
D/	07/19/1991
D2/	07/19/91

D-	07-19-1991
DY	1991
DQ	3
DD	19
DM	7
DMA	JULY

Formatting Times (MT Code)

The MT code converts times to stored format and upon output to 12-hour or 24-hour format. Like dates, times are stored in an internal format that is efficient for storage and processing. The Pick system stores times as the number of seconds after midnight (for example, 19800 is 5:30 AM).

The syntax of the MT code is as follows:

MT [H] [S]

H specifies 12-hour format with AM or PM appended. When specifying 12-hour format, input should be entered with AM or PM immediately following; if they are not specified, AM is assumed. If H is not specified, 24-hour format is assumed.

S specifies that seconds are to be included.

Table 8-14 shows some sample uses of the MT code.

Table 8-14. Time Formats.

Input Value	MT Conv	Stored Value	Output
11	MT	39600	11:00
11	MTH	39600	11:00AM
11	MTS	39600	11:00:00
11	MTHS	39600	11:00:00AM
11:15AM	MT	40500	11:15
11:15AM	MTH	40500	11:15AM
11:15PM	MT	40500	11:15

Input Value	MT Conv	Stored Value	Output
11:15PM	MTH	83700	11:15PM
1	MT	3600	01:00
1	MTH	3600	01:00AM
1PM	MT	3600	01:00
1PM	MTH	46800	01:00PM
13	MT	46800	13:00
13	MTH	46800	01:00PM

Formatting Decimal Numbers (ML and MR Codes)

The ML and MR codes allow special processing and formatting of numbers and dollar amounts. ML left-justifies the data within the masking field, MR right-justifies the data. Note that the justification within the masking field is distinct from the column justification specified by line 9 of an Attribute Definition item.

The ML and MR codes provide capabilities such as:

- Placement of commas to indicate thousands, millions, etc.
- Placing the decimal point properly and rounding off the result.
- Adding a dollar sign at the beginning of the number.
- Adding the letters "CR" (credit) after negative numbers and the letters "DB" (debit) after positive numbers.

For example, Figure 8-11 shows the elements in the conversion code MR24$. "R" right-justifies the column of figures within the masked field, "2" specifies that two digits are to be printed after the decimal point, and "4" moves the decimal point four places to the left. Since only two decimal places are to be displayed on output, the amounts are rounded off. A dollar sign is added immediately in front of each amount.

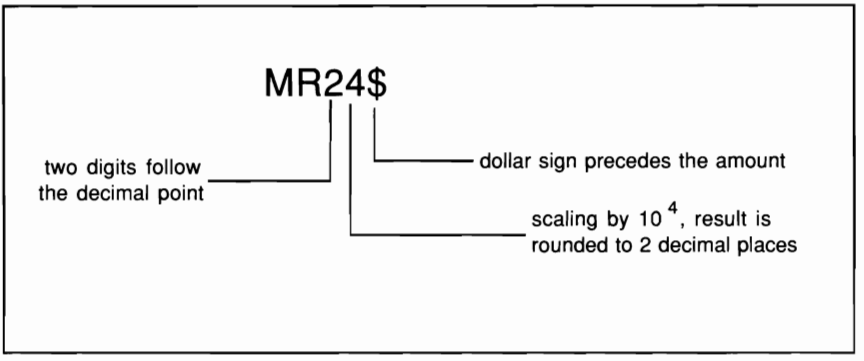


Figure 8-11. The MR Code.
The MR Code converts decimal numbers to output format.

When MR24$ is applied to an attribute containing the unformatted dollar amounts, they are listed in ACCESS reports in the following output format:

Stored Format	External Format
7270492	$727.05
375026	$37.50
13011344	$1301.13
489975	$49.00
3737373	$373.74
9294239	$929.42
459626	$45.96
199041	$19.90
2688098	$268.81
2365033	$236.50
1960013	$196.00
1125024	$112.50
497488	$49.75

The ML and MR codes use the following syntax:

M { **L** | **R** } [*n* [*m*]] [**Z**] [,] [*option*] [**$**] [(*format-mask*)]

n specifies the number of digits to be printed to the right of the decimal point. If *n* is omitted or is zero, no decimal point is printed.

m specifies that the decimal point is to be moved *m* places to the left on output. If *m* is not specified, *m* = *n* is assumed.

Z indicates that a data value of zero is to be output as null.

, specifies that commas are to be inserted every three digits to the left of the decimal point.

$ places a dollar sign immediately in front of the first digit of the number.

Table 8-15 lists the options available with the ML and MR codes.

Table 8-15. ML and MR Code Options.

Option	Description
C	adds the letters "CR" after negative values.
D	adds the letters "DB" after positive values.
E	encloses negative numbers in angle brackets (< >).
M	adds a minus sign (–) after negative values.
N	suppresses the minus sign on negative numbers.

Table 8-16 lists the format codes available with the ML and MR codes.

Table 8-16. ML and MR Format Mask Codes.

Code	Description
$	places a dollar sign at the beginning of the output field rather than immediately in front of the number.
#*n*	specifies that data is to be justified in a column of *n* blanks. If *n* is not specified, only one column is displayed.
**n*	specifies that data is to be justified in a column of *n* asterisks. If *n* is not specified, only one column is displayed.

%n	specifies that data is to be justified in a column of *n* zeros. If *n* is not specified, only one column is displayed.

Literal strings enclosed in parentheses can also be included in the format mask. The text is printed as specified, with the number being processed right- or left-justified.

Table 8-17 contains some sample ML and MR codes.

Table 8-17. Some Decimal Number Formats.

Data	Code	Output
5824	MR2	58.24
5824	MR20	5824.00
5824	MR2$	$58.24
5824	MR2D$	$58.24DB
–5824	MR2E	<58.24>
7270492	MR24$	$727.05
7270492	MR2,$	$72,704.92
5824133	MR2,$(*14)	****$58,241.33
5824133	ML(###-##-###)	582-41-33
5824133	MR(###-##-###)	58-24-133

Using Masked Character Codes (MC Codes)

Most of the character masking codes extract certain classes of characters from a string. Alphabetic, nonalphabetic, numeric, or nonnumeric characters can be extracted. Spaces are treated as non-alphanumeric characters. Other MC codes convert uppercase data to lowercase and vice versa.

Table 8-18 lists the Masked Character codes.

Table 8-18. Masked Character Codes.

Code	Description
MCA	extracts all alphabetic characters, both uppercase and lowercase. Nonalphabetic characters, including spaces, are not printed.
MC/A	extracts all nonalphabetic characters, including spaces. Alphabetic characters are not printed.
MCD[X]	converts numeric data from decimal format to its hexadecimal equivalent.
MCL	converts all uppercase letters to lowercase. Does not affect lowercase letters or nonalphabetic characters.
MCN	extracts all numeric characters. Nonnumeric characters, including spaces, are not printed.
MC/N	extracts all nonnumeric characters, including spaces. Numeric characters are not printed.
MCP	converts nonprintable characters to dots and drops the high-order bit of all characters above character 127.
MCT	converts the first character of each word to uppercase and all other characters to lowercase. Does not affect nonalphabetic characters. A word begins after any nonalphabetic character except a single or double quote.
MCU	converts all lowercase letters to uppercase. Does not affect uppercase letters or nonalphabetic characters.
MCX[D]	converts numeric data from hexadecimal format to its decimal equivalent.

Table 8-19 shows some examples of the use of these codes. A space is represented by a ◊.

Table 8-19. Masked Character Examples.

Data	Code	Output
*Boston,◊MA◊◊02134	MCA	BostonMA
*Boston,◊MA◊◊02134	MC/A	*,◊◊◊02134
*Boston,◊MA◊◊02134	MCL	*boston,◊ma◊◊02134

*Boston,◊MA◊◊02134	MCN	02134
*Boston,◊MA◊◊02134	MC/N	*Boston,◊MA◊◊
*Boston,◊MA◊◊02134	MCT	*Boston,◊Ma◊◊02134
*Boston,◊MA◊◊02134	MCU	*BOSTON,◊MA◊◊02134

Converting Hexadecimal Numbers (MCDX and MCXD)

The MCDX and MCXD codes convert data from hexadecimal to decimal or decimal to hexadecimal equivalents. When specified in line 7 as a conversion, MCXD applies a decimal-to-hexadecimal conversion and MCDX applies a hexadecimal-to-decimal conversion. When specified in line 8 as a correlative, each code applies an inverse conversion. In this case, MCXD produces a decimal equivalent and MCDX produces a hexadecimal equivalent.

The examples in Table 8-20 assume that both codes are placed in line 7 to translate ASCII character codes to decimal or hexadecimal equivalents.

Table 8-20. The MCDX and MCXD Codes.

ASCII Character	MCDX (Decimal)	MCXD (Hexadecimal)
A	65	41
B	66	42
M	77	4D
N	78	4E
Y	89	59
Z	90	5A

Converting Hexadecimal and ASCII Strings

The MX code converts a character string to its hexadecimal ASCII equivalent; the MY code converts a hexadecimal string to its alphanumeric equivalent.

The MX code converts each character to a two byte hexadecimal number. This code is useful for finding nonprintable characters in strings of data.

Table 8-21 shows some examples of the MX and MY conversions. In the first example, the string begins with a CTRL-A, which is converted to its ASCII equivalent (01).

Table. 8-21. The MX and MY Codes.

Stored Data	Code	Result of Conversion
CTRL-AABC	MX	01414243
CTRL-\	MX	FC
CTRL-]	MX	FD
CTRL-^	MX	FE
JOHN	MX	4A4F484E
john	MX	6A6F686E
414243	MY	ABC
4A4F484E	MY	JOHN
FC	MY	CTRL-\

Advanced Topics

The following sections describe three specialized uses of correlatives and conversions:

- Putting codes normally used as correlatives in line 7 (instead of line 8), and putting codes normally used as conversions in line 8 (instead of line 7).
- Using multiple correlatives and conversions in an Attribute Definition item.
- Combining correlatives and conversions in line 8 of an Attribute Definition item.

Using Correlatives as Conversions and Vice Versa

As was mentioned earlier in this chapter, codes placed in line 8 of an Attribute Definition item are correlatives, and codes placed in line 7 are conversions. Most of the codes discussed earlier in this chapter are used primarily as one or the other; for example, the D code and the masking codes (MR, ML, MX, etc.) are normally used as conversions, and the A and F codes are normally used as correlatives.

There are three main exceptions to this general practice: when using counter operands, when applying a conversion to the data *before* a correlative operation is carried out, and when applying a conversion to the data, or to a portion of the data, part of the way through the application of the correlative.

Using Special Operands with A and F Codes

When the special operands NB (number breaks), ND (number detail lines), or NI (number items) are used, or when the LPV (load previous value) operator is used with the A or the F codes, they can sometimes be used as conversions and placed in line 7. The LPV operator loads the value obtained from a previous correlative operation.

Table 8-22 shows the special operands that can be used with both the A and the F codes.

Table 8-22. Special Operands.

Operand	Description
NB	Break-level counter. Used only as a conversion in line 7.
ND	Detail line counter. Used only as a conversion in line 7.
NI	Item counter. Can be used either as a correlative in line 8 or as a conversion in line 7.
NV	Value counter. Used only as a correlative in line 8.
NS	Subvalue counter. Used only as a correlative in line 8.

The NB operand works only with ACCESS statements that contain the BREAK-ON modifier. NB returns the current break level. The lowest break level has a value of 1, a grand-total line has a value of 255.

The ND operand returns the number of detail lines since the last control break. On a detail line it has a value of 1, on a grand-total line it has the value of the item counter. ND can be used to calculate averages in conjunction with control breaks.

The NI operand returns the number of items selected or listed.

NV and NS return the number of values or subvalues (respectively) contained in the item being processed. NV and NS are for columnar listing only.

The LPV (Load Previous Value) operator takes the result of a previous correlative operation and uses it for further processing. The code with the LPV operator is specified in line 7, and the correlative that generates the "previous value" is specified in line 8. The previous value becomes an *operand* in an A code and an *element* in an F code; it must be the first element specified in an F code, otherwise strange results may be obtained.

In the following example, the F code in line 8 first multiplies the contents of Attribute 2 by the contents of Attribute 3. Then the F code in line 7 divides the result by 100.

```
007 F ; LPV ; "100" ; /
008 F ; 2 ; 3 ; *
```

Applying Conversion Operations Before Correlative Operations

If you want to apply a code that is normally used as a conversion to the data *before* a correlative operation is carried out, place the conversion code in line 8 before the correlative code. Separate the codes with a value mark (CTRL-]). This way the correlative will be applied to formatted, not raw, data.

For example, a D code could first convert a date to external format. Then a G correlative could extract one or more fields from within the delimiters defined for the date by the D conversion.

Using Multiple Codes

Two or more codes can be entered on the same line of an Attribute Definition item in one of two ways:

1. The codes can be separated with value marks (CTRL-]).
2. One code can be embedded within another code.

For example, multiple file translation correlatives make it possible to retrieve data from target files that are at two or more removes from the source file. Thus, the data for the attribute BAL.DUE in the ORDERS file can be obtained from the OVERDUE.ORDERS file by creating the following synonym for Attribute 3, CUST.ID, in the ORDERS file:

```
    BAL.DUE
001 S
002 3
003 Balance Due
004
005
006
007
008 TCUSTOMERS ; X ; ; 12 ] TOVERDUE.ORDERS ; X ; ; 6
009 R
010 8
```

The two T*file* correlatives in the preceding example reference:

- Attribute 12, BAL.DUE, of the CUSTOMERS file.
- Attribute 6, BAL.DUE, of the OVERDUE.ORDERS file.

The Attribute Definition item for Attribute 12, BAL.DUE, in the CUSTOMERS file must itself contain a T*file* correlative that points to Attribute 6, BAL.DUE, in the OVERDUE.ORDERS file:

```
    BAL.DUE
001 S
002 12
003 Balance Due
004
005
006
007
008 TOVERDUE.ORDERS ; X ; ; 6
009 R
010 8
```

This link enables the data in Attribute 6 of the OVERDUE.ORDERS file to be accessed from the ORDERS file via the two file translation correlatives (that is, through the CUSTOMERS file).

The following Attribute Definition item contains a T*file* correlative embedded in an F correlative:

```
    TAX
001 S
002 0
003 Tax
004
005
006
007 MR22
008 F ; 1 ; (TBOOKS ; X ; ; 3) ; 2 ; * ; C5 ; * ; S
009 R
010 8
```

The embedded T*file* correlative is enclosed in a set of parentheses.

In the preceding example, the F correlative uses Attribute 1, BOOKCODE, in the ORDERS file to reference the item IDs in the BOOKS file. The T*file* correlative translates data from Attribute 3, PRICE, of the BOOKS file. PRICE is then multiplied by the data in Attribute 2, QTY, of the ORDERS file, and the result is then multiplied by 5 to obtain the 5% sales tax. The S function sums the data in the multivalued attribute TAX. Finally, a masked decimal conversion (line 7) is applied to the result, which places the decimal point properly.

Combining Correlatives and Conversions

Under certain circumstances, codes normally used as conversions can be incorporated into a correlative in line 8 of an Attribute Definition item. These conversion codes are often MR or ML conversions used to format decimal numbers.

There are two main reasons for applying a conversion within a correlative:

- Ensuring that a division operation produces a correct result.
- Formatting data part of the way through the application of a correlative code. This technique is often used to convert data to output format before continuing with a series of arithmetic operations.

The following sections illustrate these two scenarios.

Adjusting Division Operations

When division is performed with the A or F correlatives, the result is always an integer. Remainders are suppressed. Therefore, to ensure that a result is returned with a remainder, scale the dividend by a multiple of 10 before dividing it, then apply a masked decimal code to insert the decimal point properly.

In the following example, the A correlative first divides the contents of Attribute 4 by 8. The MR2 conversion then formats the result:

```
007 MR2
008 A ; ( 4 * "100" ) / "8"
```

APPENDIXES

APPENDIX A

ACCESS Commands

Appendix A is a reference guide to all ACCESS verbs arranged in alphabetical order. Each entry comprises a brief explanation of what the command does, a complete explanation of its syntax, and a description of how to use the command. Commands not included in the SMA standards are marked with an asterisk.

CHECK-SUM: Produces check-sum statistics for file items.

The CHECK-SUM verb provides statistical information on all or specified items in a file. This information includes the total number of bytes, the average number of bytes, the number of items check-summed, the check-sum, and the bit count. Check-sums are useful for determining whether data in a file has been changed.

CHECK-SUM [**DICT**] *filename* [*items*] [*selection*]
[*attribute-name*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of item IDs or an item selection expression. If you do not specify *items*, all items are check-summed.

selection	specifies one or more conditions data in an item must meet to be included. For a complete description of selection expression syntax, see the LIST verb.
attribute-name	specifies one particular attribute. If you do not specify an attribute, the item IDs are check-summed.
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

To use check-sums, run CHECK-SUM on all files or portions of files to be verified, and keep a copy of the output statistics. Whenever CHECK-SUM is subsequently run, compare the new check-sum to the old one. If the check-sums are identical, the items have not changed. Remember to rerun the CHECK-SUM verb whenever the file is changed in any way.

The check-sum is calculated by multiplying the binary value of each character by its positional value. The product has a high probability of being unique for any given character string. The check-sum is the arithmetic total, minus overflow, of all bytes in the selected items.

When you run CHECK-SUM , its output is displayed in the following format:

```
BYTE STATISTICS FOR filename | attribute
TOTAL = bytes AVERAGE = avg-bytes ITEMS = items CKSUM = cksum BITS = count
```

The CHECKSUM display provides the following information based on the items processed:

filename | *attribute*	The name of the file or attribute for which the statistics are produced.
bytes	The total number of bytes for the element stored in the attribute or item.

avg-bytes	The average number of bytes.
count	The number of items check-summed.
cksum	The check-sum.
count	The bit count for the elements stored in the attribute or item.

COPY-LIST: Copies a saved select-list.

The COPY-LIST verb copies a select-list that was previously saved with the SAVE-LIST verb. You can copy the select-list to the terminal screen, to another select-list in the POINTER-FILE, or to an item in a file. COPY-LIST can also be used to copy select-lists from other accounts.

COPY-LIST *list-name* [*account*] [(*options*)]

list-name	is the name of the select-list.
account	is the name of the account containing the select-list.

options modify the copy operation. The available options are:

D	Copies the select-list to another select-list or to a file item and then deletes the original select-list. This option can be used to change a select-list's name.
N	Disables paging when copying the select-list to the terminal.
O	Overwrites a select-list if it already exists.
P	Copies the select-list to the printer.
T	Copies the select-list to the screen.
X	Copies the select-list to the printer or terminal in hexadecimal format.

To print the copied select-list, use the P option; to display it on the terminal screen, use the T option. If neither option is specified, the system prompts you as follows:

TO:

COPY-LIST

Press the RETURN key to copy the select-list to the screen. To copy the select-list to another select-list or to an item in a file, respond to the destination prompt using the following syntax:

{ *dest-list* | [(*filename*)] *item-ID* }

dest-list	is the name of the select-list to which the select-list will be copied.
filename	is the name of the file. Each element in the select-list will be stored as a separate attribute in the file item. The file must already exist.
item-ID	is the name of the new item that will contain the copied select-list.

☞ If the saved list is larger than 32K, you cannot copy it to a file item. In this case the following message is displayed:

[A96] '*list*' IS TOO LARGE TO BE AN ITEM.

COUNT: Counts file items.

The COUNT verb counts the number of items in a file that meet specified conditions. This verb summarizes the frequency with which a specific data element occurs in a database. For example, you might want to count the number of customers who live in a certain city.

COUNT [**DICT**] *filename* [*items*] [*selection*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be selected. For a complete description of selection expression syntax, see the LIST verb.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

☞ Do not confuse the COUNT verb with the PICK/BASIC COUNT function.

EDIT-LIST: Edits a select-list.

The EDIT-LIST verb invokes the Editor on a select-list that was previously saved with the SAVE-LIST verb. The select-list must be smaller than 32K.

EDIT-LIST *list-name*

list-name is the name of the select-list.

Each element in a select-list appears as a line in the Editor.

FILE-TEST: Tests item distribution in a file.

Use FILE-TEST to summarize the distribution of items in a file and to experiment with different modulos. FILE-TEST is an SMA standard verb that combines the functions of the HASH-TEST and ISTAT verbs. FILE-TEST can be used to analyze the structure of groups within a file and produce a file-hashing histogram that is helpful for determining whether the current file structure is the best one. By testing different modulos you can summarize what the item distribution in a file would be if its modulo were changed.

FILE-TEST [**DICT**] *filename* [*items*] [*selection*] [*modifiers*] [(*options*)]

DICT specifies the file dictionary.

filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

After you enter FILE-TEST, you are prompted to enter either a test modulo or a test modulo and separation. If you press the RETURN key at the prompt, the current modulo (or modulo and separation) are used.

If you enter a test parameter, FILE-TEST first analyzes what the structure of groups within a file would be, then produces a hypothetical file-hashing histogram that is helpful for determining whether the current file structure is the best one. The file against which you run FILE-TEST remains unchanged.

If you do not enter a test parameter, FILE-TEST analyzes the current structure of the file.

FORM-LIST: Selects attribute values from selected file items.

The FORM-LIST verb creates a temporary select-list containing all values either in a specified attribute or in all attributes from all or selected file items. Each attribute value becomes an element in the select-list. A subsequent command can then use the data in the select-list to reference item IDs in another file.

FORM-LIST [**DICT**] *filename* [*items*] [(*attr#*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is a list of individual item IDs or an item selection expression. Enclose each item ID in single quotes. Use an asterisk (*) to select all items in *filename*.
attr#	is the attribute number (AMC) of the attribute whose data elements will be placed in the select-list. If *attr#* is not specified, FORM-LIST selects all the attributes.

FORM-LIST creates a select-list containing data elements from the items specified in *items*. The items referenced by the select-list will be processed by the next verb you execute. For instance, you might use FORM-LIST to create a select-list of customer IDs that are stored in Attribute 4 of the ORDERS file, in order to reference names and addresses in the CUSTOMERS file.

☞ Only the statement immediately following the FORM-LIST statement will have access to the select-list. In other words, you must use the select-list immediately, or you lose it!

To permanently save the select-list, use the SAVE-LIST verb. Once a select-list is saved, you can retrieve it at any time with the GET-LIST verb.

A select-list can reference data in any file, not just the file specified in the original FORM-LIST statement. If two files have similar items with the same item IDs, you can create a select-list from one file, then use it to operate on items from the other file.

FORMS: Lists items on forms.

Not included in the SMA standards. FORMS is a forms generation verb and is not available on all systems. The FORMS verb prints file items on such forms as invoices, checks, and order forms. Using FORMS allows you to explicitly position data either on the terminal or on a printer page according to *x*- (column) and *y*- (row) coordinates. FORMS prints one item per form.

Ultimate systems use a modified version of the FORMS verb syntax. The FORMS verb itself is not supported by Ultimate; instead, a forms generation expression can be included in the LIST verb syntax. Forms generation expressions have essentially the same syntax as they do in the FORMS verb, except that the verb used is LIST instead of FORMS. Other differences in usage are noted in the following pages.

FORMS [**DICT**] *filename* [*Items*] [*selection*] *output* [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be listed on a form. For a complete description of selection expression syntax, see the LIST verb.
output	is the list of attributes to be output on the form. Each attribute specified to be output must be associated with a print code. All forms generation statements must contain at least one print code. Print codes are described below in the section "Print Codes."
	Print limiter output specifications specify which values from multivalued attributes are to be included in the report. Use relational operators and values immediately following the name of the

multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that modify the appearance of the form. These parameters affect headings, footings, and more. For complete information about modifiers, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

The following modifiers behave somewhat differently when used with forms generation verbs: BREAK-ON, FOOTING, HDR-SUPP, HEADING, and ID-SUPP. These modifiers are described in the section "Forms Generation Modifiers."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

In addition to the standard ACCESS options, the following options are also available:

A allows you to check printer alignment.

B prints a prestored "background" form along with the data specified in the FORMS statement.

M specifies the number of lines to be listed for each subpage. Not available on all systems.

Z on multipage forms, resets footing page number to one.

These nonstandard options are described in the section "Forms Generation Options."

FORMS applies the format specified in the command line to the *output* specifications. FORMS prints one item per form; a form can be either single-page or multipage. Data can be printed anywhere on a page, and can be printed on just one page or on every page of a form. In addition to specifying how the data is to be printed on a form, FORMS can be used to specify the layout of the form itself.

The @W and @D print codes (described in the next section, "Print Codes") make it possible to define vertical windows in which the data for multivalued attributes can be output. If the data in a multivalued attribute does not fit in a window, the remaining data is printed on the next page. A form can contain up to six separate windows.

Ultimate systems do not support the @W and @D codes. Their functions are implemented by the WINDOW keyword. See Chapter 7 for more information.

The SFORMS verb prints items or forms in sorted order. The REPT and SREPT verbs can be used to print more than one item on a page. On systems that do not support REPT and SREPT, the M option can be used to specify the number of items to be printed per page.

Print Codes

Each attribute name included on a form must be associated with a print code as follows:

@code (*x*,*y* [,*z*]) : *attribute-name* [**1**,*n*]

To define a character string that prints in a specified location on the form, use the following syntax:

@code (*x*,*y* [,*z*]) : "*string*" [**1**,*n*]

code	is the print code associated with the attribute. The available print codes are shown in table A-1.
x	is the horizontal position (column) on the page where the data begins. The leftmost position is column 0.

y	is the vertical position (row) on the page where the data begins. The top row is row 0, which is reserved for the heading.
z	is extra data required by the @D and @W print codes.
attribute-name	is the name of the attribute whose data is printed at the specified position.
string	is a character string that is printed at the specified position.
n	prints only the first *n* characters of the data for this attribute.

Table A-1 summarizes the available print codes.

Table A-1. Print Codes.

Print Code	Description
@[A](*x*,*y*)	Prints the data for an attribute on every page of a multipage form. The "A" is optional.
@C(*x*,*y*)	Creates an audit trail for a series of forms. The @C code can also be used to serialize the forms. To suppress serialization numbers on the form (but not in the audit file), specify (–1) in place of the *x* and *y* coordinates. (You *must* use either the *x*-*y* coordinates or –1 with the @C code.) When you use the @C print code, the system prompts for the audit file and starting number: AUDIT FILE> Enter the name of an existing file that will be used to track each form that is generated. STARTING NUMBER> Enter the starting serialization number for the forms. This number will appear on the form at the position specified for the @C-formatted attribute. The information on each form will be stored as an item in the audit file. See Chapter 7, "Forms Generation," for more information about audit trails.

@D(*x*,*y*,*z*)	Not available on all systems. The @D print code prints data for multivalued attributes in double-depth windows. This makes it possible to define two lines of output at a time. *z* specifies the bottommost row of a window whose topmost row is defined by *y*. See also the WINDOW keyword. You must add an S to the end of the *x* parameter for an attribute to appear on every second output line.
@F(*x*,*y*)	Prints the data for an attribute on only the first page of a multipage form.
@L(*x*,*y*)	Prints the data for an attribute on only the last page of a multipage form.
@M(*x*,*y*,"*text*")	Prints the specified text on all but the last page of a multipage form. The data for the attribute prints on the last page of the form. On a single-page form, the data for the attribute is printed.
@W(*x*,*y*,*z*)	Not available on all systems. The @W print code prints data for multivalued attributes in windows. *x* specifies the columns where attribute values are to start being printed. *z* specifies the bottommost row of a window whose topmost row is defined by *y*. See also the WINDOW keyword.

The system does not check whether output data exceeds the page width set by the TERM verb. The justification (V/TYP) of the specified attribute in combination with the maximum number of characters (V/MAX) and the column position (specified by *x*) determine the width of the data displayed on the form.

For complete information about using print codes, see Chapter 7, "Forms Generation.

Forms Generation Modifiers

When used with forms generation verbs, the following modifiers behave somewhat differently than they do with other ACCESS verbs. For more information about how these modifiers work with forms generation verbs, see Chapter 7, "Forms Generation."

BREAK-ON

The data line for the BREAK-ON modifier is treated like any other *output* specification when it is printed.

Generally, forms generation verbs will not use the BREAK-ON, TOTAL, and GRAND-TOTAL modifiers.

HEADING and FOOTING

Headings and footings cannot be output on a form using *x-y* coordinates; they are output in separate reserved areas. Conversely, data specified by the *output* parameter cannot be placed in the areas reserved for headings and footings. By default, row 0 is reserved for headings and row 1 is reserved for a blank line that separates the heading from the report. If a multiple-line heading is specified for either headings or footings, the system reserves the number of lines needed for the text. The blank line is replaced by the specified multiple-line heading.

HDR-SUPP

Forms generation verbs automatically suppress column headings, but they still generate the one-line heading that lists the time, date, and page number. HDR-SUPP suppresses this one-line heading.

ID-SUPP

Forms generation verbs automatically suppress item IDs, so this modifier is unnecessary in a forms generation statement. If you want to include item IDs on a form, create an item in the dictionary that has an attribute number (AMC) of 0. Then include an *output* specification referring to this item in the forms generation statement.

Forms Generation Options

The following nonstandard options can be specified with FORMS: A, B, M and Z.

The A Option

The A option is used to verify the page layout before the actual forms are generated. The A option runs a printer-alignment routine. This routine prints the first form on the terminal screen or printer, showing text (except for headings and footings) as Xs. This option can be used only if the current SP-ASSIGN statement includes the C and the I options.

The following prompt appears if the A option is included in the forms generation statement:

```
Align? Y=cr/N>
```

Before you respond to this prompt, manually set the printer at the top of the form. Then press the RETURN key to display or print the data as Xs. Repeat this process as many times as you need to until you get the Xs to print where you want them. Then enter "N" at the prompt to print the forms themselves.

The B Option

B prints a background form composed of previously created text or graphics. It is printed on every page of a form. To use the B option, you must already have created a background form (background forms are created with the Editor). This background form can be used to print forms on standard paper instead of on preprinted forms. Background forms also make it possible to use printer control characters (for underlining, etc.) that apply to specific printers.

Design a background form so that it is not overwritten by the reserved heading rows 0 and 1, and does not overwrite the data placed on the form with print codes.

The following prompt appears if the B option is included in the forms generation statement:

Background File & Item:

Enter the name of the background file item using the following format:

[**DICT**] *filename item-ID*

The M Option

On systems that do not have the REPT and SREPT verbs, the M option is used to specify the number of lines each subpage should include. After FORMS with the M option is entered, the system prompts for subpage size:

Subpage size>

Enter the number of lines each subpage should contain. For example, to define 6 subpages on each page of 62 lines (2 lines are reserved for the heading), enter 10. Each subpage extends the full width of the page.

The system does not split items; if an entire subpage will not fit at the bottom of a page, the system prints the subpage on the next page.

See the REPT verb and Chapter 7 for information about subpages.

The Z Option

Z is used with the FOOTING modifier. It resets the page number in the footing back to 1 at the beginning of each multipage form. Use the HDR-SUPP modifier with the Z option to prevent an incompatible (not reset) page number from appearing in the heading.

GET-LIST: Retrieves a previously saved select-list.

The GET-LIST verb retrieves a select-list that was previously saved with the SAVE-LIST verb. After retrieval you can execute a single command

on the specified select-list. To execute more than one command, you must retrieve the select-list again.

GET-LIST [[**DICT**] *filename*] *list-name*

DICT	is the file dictionary.
filename	is the file where the select-list is stored.
list-name	is the name of the select-list that you want to retrieve.

You can retrieve any select-list created from another account if it was saved in the system-level POINTER-FILE.

*HASH-TEST: Tests effects of different modulos on item distribution.

Not included in the SMA standards. Use HASH-TEST to experiment with different modulos and to produce a summary of what the item distribution in a file would be if its modulo were changed. This command can be used to determine what modulo would produce the best item distribution for a file. HASH-TEST has been superseded by the FILE-TEST verb.

HASH-TEST [**DICT**] *filename* [*items*] [*selection*] [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete

information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

HASH-TEST first analyzes what the structure of groups within a file would be, then produces a hypothetical file-hashing histogram that is helpful for determining whether the current file structure is the best one. The file against which you run HASH-TEST remains unchanged.

Once you have determined what the new modulo for the file should be, you can reallocate the file by entering the new modulo in Attribute 13 of the File Definition item. The file will be reallocated automatically after the next file-save and file-restore.

☞ To reallocate a data file, change Attribute 13 in the D-pointer located in the file dictionary. To reallocate a file dictionary, change Attribute 13 in the D-pointer located in the Master Dictionary of the account. To reallocate an account's Master Dictionary, change Attribute 13 in the SYSTEM Dictionary.

*ISTAT: Summarizes item distribution in a file.

Not included in the SMA standards. The ISTAT verb summarizes the distribution of items in a file. ISTAT analyzes the structure of groups within a file and produces a file-hashing histogram that is helpful for determining whether the current file structure is the best one. ISTAT has been superseded by the FILE-TEST verb.

ISTAT [**DICT**] *filename* [*items*] [*selection*] [*modifiers*]
[(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

LIST: Generates reports from a database.

The LIST verb produces a formatted report which can be displayed on the screen or sent to the printer. These reports can display items from either the data file or the file dictionary. LIST is one of the most frequently used ACCESS verbs.

LIST [*file-modifiers*] *filename* [*items*] [*selection*] [*output*] [*modifiers*] [**(** *options* **)**]

file-modifiers can be DICT or ONLY. DICT specifies the file dictionary. ONLY suppresses the default output specification and displays item IDs only.

filename is the name of the file.

items is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.

selection specifies one or more conditions that data in an item must meet to be included in the report. *selection* has this syntax:

WITH [**EACH**] *attribute-name* [[*rel-op*] *value-list*] [{ **AND** | **OR** } **WITH** [**EACH**] *attribute-name* [[*rel-op*] *value-list*]]

EACH specifies that all values in a multivalued attribute must meet the specified condition if the item is to be included in the report.

attribute-name is the name of the attribute whose data values are to be compared to the specified condition.

rel-op can be any relational operator. For a complete list of relational operators, see Chapter 4, "Formatting Reports."

value-list can be either one or more data values, or a constant. Values should be enclosed in double quotes.

AND | OR specifies a compound expression.

output is a list of the names of one or more attributes whose data is to be included in the report. *output* can also be a user-defined phrase that contains any ACCESS parameters except a verb or a filename.

Print limiting output specifications specify which values from multivalued attributes are to be listed in the report. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

LIST displays information in either columnar or noncolumnar format. In addition, unless you use the ID-SUPP modifier, this command automatically displays item IDs as the first column in the display.

SORT is another of the most frequently used ACCESS verbs. SORT generates reports similar to those produced by LIST , and also sorts items in alphabetical or numeric order.

LIST-ITEM: Displays all data for items.

The LIST-ITEM verb displays all attribute values for specified file items. This verb is useful for producing a simple listing of data or dictionary items. LIST-ITEM combines the function of the COPY processor with the ability of ACCESS to select specified items.

LIST-ITEM [**DICT**] *filename* [*items*] [*selection*] [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

If no items are specified or selected, all items in the file are listed. By default, LIST-ITEM displays items on the screen. The display includes line numbers for each attribute, unless the S option is used.

LIST-LABEL: Lists data in label format.

The LIST-LABEL verb allows you to specify a format for specialized block listings such as mailing labels. LIST-LABEL can be used to define how many blocks (or items) are displayed across each page or screen and how many rows (or attributes) are displayed for each block. LIST-LABEL also defines the number of vertical lines and horizontal spaces between blocks, the amount of indent from the left margin of the page or screen, and the maximum width of a row.

LIST-LABEL [**DICT**] *filename* [*items*] [*selection*] [*output*] [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included. For a complete description of selection expression syntax, see the LIST verb.
output	is the list of attributes to be included in the labels. *output* can also be a user-defined phrase that contains any ACCESS parameters except a verb or a filename. Print limiting output specifications specify which values from multivalued attributes are to be included. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as

commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

After LIST-LABEL is entered, the system displays the following prompt:

?

You can now determine the format of the label. Enter a response in the following format:

count, rows, skip, indent, size, space [**,C**]

count	is the number of labels (items) across each page or screen.
rows	is the number of lines printed for each label. Remember to count the item ID as one line. The item ID is automatically included in the labels unless you use the ID-SUPP modifier or the I option in the query.
skip	is the number of lines to skip vertically between labels.
indent	is the number of indented spaces from the left margin to the label. Zero (0) is a valid response.
size	is the maximum width for the data contained in each attribute (in other words, the width of each label in columns).
space	is the number of horizontal spaces between labels.
C	specifies that null attributes should not be printed. Otherwise, null attributes appear as all blanks. This parameter is optional.

The *size* parameter cannot exceed the page width (80 characters for terminals, and 80/132 characters for printers). Calculate label width as follows:

(*count* * (*size* + *space*) + *indent*) <= (*current page width*)

where *current page width* is the value defined by the TERM or SET-TERM verbs for the terminal or printer.

If a value other than zero is specified for the *indent* parameter, the system prompts you to define headers for each row in a label:

```
?
```

The number of ? prompts corresponds to the value entered earlier for *rows*. At each ? prompt, enter the desired header. To avoid defining a header, simply press the RETURN key. Defined headers will appear at the left margin in the *indent* area of the listing.

*NSELECT: Selects items from an active select-list that aren't in a file.

Not included in the SMA standards. The NSELECT verb creates a new select-list containing only those items in the current select-list that are *not* in the file specified by the NSELECT statement. In order to use NSELECT, you must have just created or retrieved a select-list with one of the following verbs: SELECT, SSELECT, QSELECT, FORM-LIST, or GET-LIST.

NSELECT *filename*

filename	is the name of the file against which the current select-list is compared.

NSELECT creates a new select-list containing item IDs of only those items in the previous select-list that are not also in *filename*. The items referenced by the new select-list will be processed by the next verb you execute.

Only the statement immediately following the NSELECT statement will have access to the select-list. In other words, you must use the select-list immediately, or you lose it!

To permanently save the select-list, use the SAVE-LIST verb. Once a select-list is saved, you can retrieve it at any time with the GET-LIST verb.

A select-list can reference data in any file, not just the file specified in the original SELECT statement. If two files have similar items with the same item IDs, you can create a select-list from one file, then use it to operate on items from the other file.

Obviously, there is no point in referencing the file specified by the NSELECT statement since NSELECT has just removed from the active select-list all item IDs contained in that file.

*QSELECT: Selects attribute values from selected file items.

Not included in the SMA standards. This verb is a synonym for the FORM-LIST verb.

REFORMAT: Restructures file items.

The REFORMAT verb restructures file items and sends output to a file or to magnetic tape.

REFORMAT [*file-modifiers*] *filename* [*items*] [*selection*] [*output*] [*modifiers*] [(*options*)]

file-modifiers	can be DICT or ONLY. DICT specifies the file dictionary. ONLY suppresses the default output specification and lists item IDs only.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be transferred. For a complete description of selection expression syntax, see the LIST verb.
output	is the list of attribute-names whose values are to be transferred. If you do not include this parameter, REFORMAT transfers only item IDs. You can also specify a phrase. Print limiting output specifications specify which values from multivalued attributes are to be transferred. Use relational operators and values

immediately following the name of the multivalued attribute.

modifiers include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

The contents of any attribute in the original file can be made the item IDs for the items in the restructured destination file. This is useful for creating a new file that consists of a subset of the attributes in an existing file. The values that are to be made into item IDs must be unique, however; a null or any other set of identical values will become the item ID of one item, and the data in those items will be stored as multivalues in the item in the destination file.

After REFORMAT is entered with accompanying parameters, the system displays the following prompt:

```
FILE NAME:
```

At this point you can perform one of three different operations:

1. Transfer items to magnetic tape. To do so, enter the word TAPE at the prompt. You must have already attached the tape unit and set the tape record length with T-ATT. Each item is written to tape as a separate physical tape record (or block). If an item is larger than the tape's block size, the next tape record is used.

2. Transfer items to another file. To do so, enter the name of an existing file as follows:

   [DICT] *filename*

The system copies the selected or specified items to the destination filename.

If you specify attributes for output and include the ID-SUPP modifier in the ACCESS query, the values in the first attribute of the output specification become the item IDs for the items copied to the destination file. The second attribute listed in the output specification becomes Attribute 1 in the destination file, the third attribute becomes Attribute 2, etc. The destination file can thus consist of a subset of the attributes defined for the original file.

If you restructure files in this way and you want to generate reports from the destination file, you could copy the Attribute Definition items from the dictionary of the original file to the dictionary of the destination file and then edit the attribute numbers to reflect the new item structure.

3. Transfer items within the same file. To do so, press the RETURN key without entering a response.

Use REFORMAT with care! Most ACCESS verbs generate reports without altering the actual data stored in a database. REFORMAT, however, is an exception, since it can create new file items and alter existing file items.

*REPT: Lists multiple items on forms.

Not included in the SMA standards. REPT is a forms generation verb and is not available on all systems. The REPT verb, like the FORMS verb, prints file items on such forms as invoices, checks, and order forms. Unlike the FORMS and SFORMS verbs, which list one item per page, REPT (and SREPT) can list multiple items as *subpages* on a single page. Using REPT allows you to explicitly position data either on the terminal or on a printer page according to *x*- (column) and *y*- (row) coordinates.

Ultimate systems do not have the REPT and SREPT verbs. Instead, subpages are implemented with the M option used with LIST or SORT in conjunction with a forms generation expression. See the FORMS and

SFORMS verbs, and Chapter 7, "Forms Generation," for more information about the M option.

REPT [**DICT**] *filename* [*items*] [*selection*] *output* [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be listed on a form. For a complete description of selection expression syntax, see the LIST verb.
output	is the list of attributes to be output on the form. Each attribute specified to be output must be associated with a print code. All forms generation statements must contain at least one print code. Print codes are described below in the section "Print Codes." Print limiting output specifications specify which values from multivalued attributes are to be included in the report. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.
modifiers	include one or more keywords that modify the appearance of the form. These parameters affect headings, footings, and more. For complete information about modifiers, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords." The following modifiers behave somewhat differently when used with forms generation verbs: BREAK-ON, FOOTING, HDR-SUPP, HEADING, and ID-SUPP. These modifiers are

described in the section "Forms Generation Modifiers."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

In addition to the standard ACCESS options, the following options are also available:

A allows you to check printer alignment.

B prints a prestored "background" form along with the data specified in the REPT statement.

Z on multipage forms, resets footing page number to one.

These nonstandard options are described in the section "Forms Generation Options."

After REPT is entered, the system prompts for subpage size:

```
Subpage size>
```

Enter the number of lines each subpage should contain. For example, to define 6 subpages on each page of 62 lines (2 lines are reserved for the heading), enter 10. Each subpage extends the full width of the page.

The system does not split items; if an entire subpage will not fit at the bottom of a page, the system prints the subpage on the next page.

REPT applies the format specified in the command line to the *output* specifications. REPT prints one item per form; a form can be either single-page or multipage. Data can be printed anywhere on a page, and can be printed on just one page or on every page of a form. In addition to specifying how the data is to be printed on a form, REPT can be used to specify the layout of the form itself.

The @W and @D print codes (described in the next section, "Print Codes") enable you to define vertical windows in which the data for multivalued attributes can be output. If the data in a multivalued attribute does not fit in a window, the remaining data is printed on the next page. A form can contain up to six separate windows.

Ultimate systems do not support the @W and @D codes. Their functions are implemented by the WINDOW keyword. See Chapter 7 for more information.

As was mentioned earlier, the FORMS and SFORMS verbs print one item per page. REPT and SREPT can be used to print more than one item on a page.

Print Codes

Each attribute name included on a form must be associated with a print code as follows:

@code (*x*,*y* [,*z*]) : *attribute-name* [**1**,*n*]

To define a character string that prints in a specified location on the form, use the following syntax:

@code (*x*,*y* [,*z*]) : "*string*" [**1**,*n*]

code	is the print code associated with the attribute. The available print codes are shown in Table A-2.
x	is the horizontal position (column) on the page where the data begins. The leftmost position is column 0.
y	is the vertical position (row) on the page where the data begins. The top row is row 0, which is reserved for the heading.
z	is extra data required by the @W and @D print codes.
attribute-name	is the name of the attribute whose data is printed at the specified position.

string	is a character string that is printed at the specified position.
n	prints only the first *n* characters of the data for this attribute.

The following table summarizes the available print codes.

Table A-2. Print Codes.

Print Code	Description
@[A](*x*,*y*)	Prints the data for an attribute on every page of a multipage form. The "A" is optional.
@C(*x*,*y*)	Creates an audit trail for a series of forms. The @C code can also be used to serialize the forms. To suppress serialization numbers on the form (but not in the audit file), specify (–1) in place of the *x*- and *y*-coordinates. (You *must* use either the *x*-*y* coordinates or –1 with the @C code.) When you use the @C print code, the system prompts for the audit file and starting number: Audit File> Enter the name of an existing file that will be used to track each form that is generated. Starting Number> Enter the starting serialization number for the forms. This number will appear on the form at the position specified for the @C-formatted attribute. The information on each form will also be stored as an item in the audit file. See Chapter 7, "Forms Generation," for more information about audit trails.
@D(*x*,*y*,*z*)	Not available on all systems. The @D print code prints data for multivalued attributes in double-depth windows so you can define two lines of output at a time. *z* specifies the bottommost row of a window whose topmost row is defined by *y*. See also the WINDOW keyword. You must add an S to the end of the *x* parameter for an attribute to appear on every second output line.

@F(*x*,*y*)	Prints the data for an attribute on only the first page of a multipage form.
@L(*x*,*y*)	Prints the data for an attribute on only the last page of a multipage form.
@M(*x*,*y*,"*text*")	Prints the specified text on all but the last page of a multipage form. The data for the attribute prints on the last page of the form. On a single-page format form, the data for the attribute is printed.
@W(*x*,*y*,*z*)	Not available on all systems. The @W print code prints data for multivalued attributes in windows. *x* specifies the columns where the attribute values are to start being printed. *z* specifies the bottommost row of a window whose topmost row is defined by *y*. See also the WINDOW keyword.

The system does not check whether output data exceeds the page width set by the TERM verb. The justification (V/TYP) of the specified attribute in combination with the maximum number of characters (V/MAX) and the column position (specified by *x*) determine the width of the data displayed on the form.

For complete information about using print codes, see Chapter 7, "Forms Generation."

Forms Generation Modifiers

When used with forms generation verbs, the following modifiers behave somewhat differently than they do with other ACCESS verbs. For more information about how these modifiers work with forms generation verbs, see Chapter 7, "Forms Generation."

BREAK-ON

The data line for the BREAK-ON modifier is treated like any other *output* specification when it is printed.

Generally, forms generation verbs will not use the BREAK-ON, TOTAL, and GRAND-TOTAL modifiers.

HEADING and FOOTING

Headings and footings cannot output on a form using *x-y* coordinates; they are output in separate reserved areas. Conversely, data specified by the *output* parameter cannot be placed in the areas reserved for headings and footings. By default, row 0 is reserved for headings and row 1 is reserved for a blank line that separates the heading from the report. If a multiple-line heading is specified for either headings or footings, the system reserves the number of lines needed for the text. The blank line is replaced by the specified multiple-line heading.

HDR-SUPP

Forms generation verbs automatically suppress column headings, but they still generate the one-line heading that lists the time, date, and page number. HDR-SUPP suppresses this one-line heading.

ID-SUPP

Forms generation verbs automatically suppress item IDs, so this modifier is unnecessary in a forms generation statement. If you want to include item IDs on a form, create an item in the dictionary that has an attribute number (AMC) of 0. Then include an *output* specification referring to this item in the forms generation statement.

Forms Generation Options

The following nonstandard options can be specified with REPT: A, B, and Z.

The A Option

The A option is used to verify the page layout before the actual forms are generated. The "A" option runs a printer-alignment routine. This routine

prints the first form on the terminal screen or printer, showing text (except for headings and footings) as Xs. This option can be used only if the current SP-ASSIGN statement includes the C and the I options. (See the SP-ASSIGN verb for more details.)

The following prompt appears if the A option is included in the forms generation statement:

```
Align? Y=cr/N>
```

Before you respond to this prompt, manually set the printer at the top of the form. Then press the RETURN key to display or print the data as Xs. Repeat this process as many times as you need until you get the Xs to print where you want them. Then enter "N" at the prompt to print the forms themselves.

The B Option

B prints a background form composed of previously created text or graphics. It is printed on every page of a form. To use the B option, you must already have created a background form (background forms are created with the Editor). This background form can be used to print forms on standard paper instead of on preprinted forms. Background forms also make it possible to use printer control characters (for underlining, etc.) that apply to specific printers.

Design a background form so that it is not overwritten by the reserved heading rows 0 and 1, and does not overwrite the data placed on the form with print codes.

The following prompt appears if the B option is included in the forms generation statement:

```
Background File & Item:
```

Enter the name of the background file item using the following format:

[**DICT**] *filename item-ID*

The Z Option

Z is used with the FOOTING modifier. It resets the page number in the footing back to 1 at the beginning of each multipage form. Use the HDR-SUPP modifier with the Z option to prevent an incompatible (not reset) page number from appearing in the heading.

S-DUMP: Copies sorted items to magnetic tape.

The S-DUMP verb transfers a copy of all or selected file items in sorted order to magnetic tape. This verb also creates a tape label and writes an End-Of-File (EOF) mark on the tape after the transfer is complete.

S-DUMP [**DICT**] *filename* [*items*] [*selection*] [*sort*]
[**HEADING** " *text* "] [*modifiers*] [(*options*)]

DICT	specifies the file dictionary. When copying dictionary items, S-DUMP does not copy any File Definition items to tape.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be copied. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort by and whether to sort items in ascending or descending order. See the SORT verb for a complete list of the modifiers used to define sort expressions.
text	is added to the standard tape label via the HEADING modifier.
modifiers	include one or more keywords that specify the report format. These parameters affect headers,

footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports."

The following modifiers operate differently with T-DUMP:

HDR-SUPP suppresses the creation of a tape label.

ID-SUPP suppresses listing of item IDs during the copy-to-tape operation.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

Before using S-DUMP, use T-ATT to attach the tape unit and set the tape record length. You can select a block length, or you can accept the system default block length of 4000 bytes for 1/2-inch tape or 16,896 bytes for 1/4-inch cartridge tape. In addition, the logical tape unit should already have been selected with the T-SELECT verb.

If the tape's write protection is on or if no write ring is present, the following message is displayed:

```
WRITE PROTECTED
CONTINUE/QUIT (C/Q)?
```

Remove the write protection or install a write ring, then enter "C" to continue, or enter "Q" to quit.

S-DUMP automatically writes a label to the tape in the following format:

[**DICT**] *filename*

You can add text to this label with the *text* parameter, or you can suppress the tape label completely by including the HDR-SUPP keyword in the S-DUMP command line.

For information about reading a tape created with S-DUMP, see the T-LOAD verb.

SAVE-LIST: Saves a select-list.

The SAVE-LIST verb names and saves a select-list that was created with one of the following verbs: SELECT, SSELECT, NSELECT, QSELECT, and FORM-LIST. Once the select-list has been saved, it can be retrieved at any time with the GET-LIST verb and used again. SAVE-LIST must be entered immediately after you create a select-list.

SAVE-LIST *list-name*

list-name specifies the name of the saved select-list.

If you have previously saved a select-list with the same name, the system overwrites the old select-list with the new one without asking for confirmation.

If a select-list is saved to the system POINTER-FILE (in the SYSPROG account), it can be retrieved for use by any user from any account.

SELECT: Selects items for further processing.

The SELECT verb creates a temporary select-list of file items to be processed by the next TCL or ACCESS statement, or by other processors such as the Editor, PICK/BASIC, or PROC. Creating a select-list is a useful way to define and operate on a subset of items in a database.

SELECT [**DICT**] *filename* [*items*] [*selection*] [*output*]

DICT specifies the file dictionary.

filename is the name of the file.

items is a list of individual item IDs or an item selection expression. Enclose each item ID in single quotes. If you do not specify *items*, SELECT selects all items in *filename*.

selection specifies one or more conditions that data in an item must meet to be included in the select-list. For a complete description of selection expression syntax, see the LIST verb.

output is the name of one or more attributes whose values are to be selected. Each value becomes a separate element in the select-list. If *output* is specified, no item IDs will be selected.

SELECT creates a temporary select-list containing item IDs or data values of items specified by *items* or the selection expression. The items referenced by the select-list will be processed by the next verb you execute. For instance, before invoking the Editor you might create a select-list as a way of specifying which items you want to edit.

☞ Only the statement immediately following the SELECT statement will have access to the select-list. In other words, you must use the select-list immediately, or you lose it!

To permanently save the select-list, use the SAVE-LIST verb. Once a select-list is saved, you can retrieve it at any time with the GET-LIST verb.

A select-list can reference data in any file, not just the file specified in the original SELECT statement. If two files have similar items with the same item IDs, you can create a select-list from one file, then use it to operate on items from the other file.

*SFORMS: Lists items on forms in sorted order.

Not included in the SMA standards. SFORMS is a forms generation verb and is not available on all systems. The SFORMS verb prints file items in sorted order on such forms as invoices, checks, and order forms. Using SFORMS allows you to explicitly position data either on the terminal or on a printer page according to *x*- (column) and *y*- (row) coordinates. SFORMS prints one item per form.

Ultimate systems use a modified expression of the SFORMS verb syntax. The SFORMS verb itself is not supported by Ultimate; instead, a *forms generation expression* can be included in the SORT verb syntax. Forms generation expressions have essentially the same syntax as they do in the SFORMS verb, except that the verb used is SORT instead of FORMS. Other differences in usage are noted in the following pages.

SFORMS [**DICT**] *filename* [*items*] [*selection*] [*sort*] *output* [*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
item-list	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be listed on a form. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort and whether to sort them in descending or ascending order. See the SORT verb for a description of the four sort modifiers.
output	is the list of attributes to be output on the form. Each attribute specified to be output must be associated with a print code. All forms generation statements must contain at least one print code. Print codes are described below in the section "Print Codes." Print limiter output specifications specify which values from multivalued attributes are to be included in the report. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that modify the appearance of the form. These parameters affect headings, footings, and more. For complete information about modifiers, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

The following modifiers behave somewhat differently when used with forms generation verbs: BREAK-ON, FOOTING, HDR-SUPP, HEADING, and ID-SUPP. These modifiers are described in the "Forms Generation Modifiers" section of the FORMS verb.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

In addition to the standard ACCESS options, the following options are also available:

A allows you to check printer alignment.

B prints a prestored "background" form along with the data specified in the SFORMS statement.

M specifies the number of lines to be listed for each subpage. Not available on all systems.

Z on multipage forms, resets footing page number to one.

These nonstandard options are fully described in the "Forms Generation Options" section of the FORMS verb.

SFORMS applies the format specified in the command line to the *output* specifications. SFORMS prints one item per form; a form can be either single-page or multipage. Data can be printed anywhere on a page, and can be printed on just one page or on every page of a form. In addition to specifying how the data is to be printed on a form, SFORMS can be used to specify the layout of the form itself.

The @W and @D print codes (described in the next section, "Print Codes") make it possible to define vertical windows in which the data for multivalued attributes can be output. If the data in a multivalued attribute does not fit in a window, the remaining data is printed on the next page. A form can contain up to six separate windows.

Ultimate systems use a modified version of the FORMS verb syntax. The FORMS verb itself is not supported by Ultimate; instead, a forms generation expression can be included in the LIST verb syntax. Forms generation expressions have essentially the same syntax as they do in the FORMS verb, except that the verb used is LIST instead of FORMS. Other differences in usage are noted in the following pages.

The REPT and SREPT verbs can be used to print more than one item on a page.

Print Codes

Each attribute name included on a form must be associated with a print code as follows:

@*code* **(** *x*,*y* [,*z*] **) :** *attribute-name* [**1**,*n*]

To define a character string that prints in a specified location on the form, use the following syntax:

@*code* **(** *x*,*y* [,*z*] **) :** "*string*" [**1**,*n*]

code	is the print code associated with the attribute. The available print codes are shown in Table A-3.
x	is the horizontal position (column) on the page where the data begins. The leftmost position is column 0.

y	is the vertical position (row) on the page where the data begins. The top row is row 0, which is reserved for the heading.
z	is extra data required by the @D and @W print codes.
attribute-name	is the name of the attribute whose data is printed at the specified position.
string	is a character string that is printed at the specified position.
n	prints only the first *n* characters of the data for this attribute.

Table A-3 summarizes the available print codes.

Table A-3. Print Codes

Print Code	Description
@[A](*x*,*y*)	Prints the data for an attribute on every page of a multipage form. The "A" is optional.
@C(*x*,*y*)	Creates an audit trail for a series of forms. The @C code can also be used to serialize the forms. To suppress serialization numbers on the form (but not in the audit file), specify (–1) in place of the *x* and *y* coordinates. (You *must* use either the *x*-*y* coordinates or –1 with the @C code.)
@D(*x*,*y*,*z*)	Not available on all systems. the @D print code prints data for multivalued attributes in double-depth windows. This makes it possible to define two lines of output at a time. *z* specifies the bottom-most row of a window whose top-most row is defined by *y*. See also the WINDOW keyword. You must add an S to the end of the *x* parameter for an attribute to appear on every second output line.
@F(*x*,*y*)	Prints the data for an attribute on only the first page of a multipage form.
@L(*x*,*y*)	Prints the data for an attribute on only the last page of a multipage form.

@M(*x*,*y*,"*text*")	Prints the specified text on all but the last page of a multipage form. The data for the attribute prints on the last page of the form. On a single-page form, the data for the attribute is printed.
@W(*x*,*y*,*z*)	Not available on all systems. The @W print code prints data for multivalued attributes in windows. *x* specifies the columns where attribute values are to start being printed. *z* specifies the bottom-most row of a window whose top-most row is defined by *y*. See also the WINDOW keyword.

For complete information about using print codes, see the FORMS verb and Chapter 7, "Forms Generation."

SORT: Generates reports in sorted order from a database.

The SORT verb produces a sorted and formatted report which can be displayed on the screen or sent to the printer. SORT produces a display almost identical to that of the LIST command, except in sorted order.

SORT [*file-modifiers*] *filename* [*items*] [*selection*] [*sort*] [*output*] [*modifiers*] [(*options*)]

file-modifiers	can be DICT or ONLY. DICT specifies the file dictionary. ONLY suppresses the default output specification and displays item IDs only.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes. The system sorts these items in the order that you specify with *sort*.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort by and whether to sort items in ascending or descending order.

You can use the following modifiers in a sort expression:

BY	sorts items in ascending order by the specified attribute.
BY-DSND	sorts items in descending order by the specified attribute.
BY-EXP	sorts a multivalued attribute in ascending order and produces a separate line for each value.
BY-EXP-DSND	sorts a multivalued attribute in descending order and produces a separate line for each value.

If more than one sort expression is specified in a SORT command line, the system sorts these attributes from left to right (the leftmost sort expression is the most significant).

output is a list of the names of one or more attributes whose data is to be included in the report. *output* can also be a user-defined phrase that contains any ACCESS parameters except a verb or filename.

Print limiting output specifications specify which values from multivalued attributes are to be listed in the report. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that specify the format of the report. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify

output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

SORT-ITEM: Displays sorted items.

The SORT-ITEM verb displays all data elements for a list of sorted items. This verb is useful for producing a sorted listing of data or dictionary items, arranged according to the data elements of a specified attribute. It is also useful for displaying programs and Procs. SORT-ITEM combines the functions of the COPY processor with the ability of ACCESS to select specified items.

SORT-ITEM [**DICT**] *filename* [*items*] [*selection*] [*sort*]
[*modifiers*] [(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the report. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort by and whether to sort items in ascending or descending order. See the SORT verb for a complete list of the modifiers used to define sort expressions.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, and more. For complete information about using these keywords, see

Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

If no items are specified or selected, all items in the file are listed. By default, the SORT-ITEM verb sorts and displays items on the screen. The display includes line numbers for each attribute, unless the S option is used.

SORT-LABEL: Lists data in label format and in sorted order.

The SORT-LABEL verb allows you to specify a format for specialized block listings such as mailing labels, and to display or print them in sorted order. SORT-LABEL can be used to define how many blocks (or items) are displayed across each page or screen and how many rows (or attributes) are displayed for each block. SORT-LABEL also defines the number of vertical lines and horizontal spaces between blocks, the amount of indent from the left margin of the page or screen, and the maximum width of a row.

SORT-LABEL [**DICT**] *filename* [*items*] [*selection*] [*sort*] [*output*] [*modifiers*] [(*options*)]

DICT specifies the file dictionary.

filename is the name of the file.

items is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes. If *items* is not specified, the SORT-LABEL sorts all items in *filename* in the order that you specify with *sort*.

selection specifies one or more conditions that data in an item must meet to be included. For a complete description of selection expression syntax, see the LIST verb.

sort specifies which attributes to sort by and whether to sort items in ascending or descending order. See the SORT verb for a complete list of the modifiers used to define sort expressions.

output is a list of attributes to be included in the labels. *output* can also be a user-defined phrase that contains any ACCESS parameters except a verb or filename.

Print limiting output specifications specify which values from multivalued attributes are to be included. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

Use the COL-HDR-SUPP modifier or the C option to suppress the header (page number, time, and date) at the top of each page of the report. This modifier also suppresses pagination and all top-of-forms, thus producing a continuous forms format without page breaks.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as

commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

After SORT-LABEL is entered, the system displays the following prompt:

?

You can now determine the format of the label. Enter a response in the following format:

count, rows, skip, indent, size, space [**,C**]

count	is the number of labels (items) across each page or screen.
row	is the number of lines printed for each label. Remember to count the item ID as one line. The item ID is automatically included in the labels unless you use the ID-SUPP modifier in the query.
skip	is the number of lines to skip vertically between labels.
indent	is the number of indented spaces from the left margin to the label. Zero (0) is a valid response.
size	is the maximum width for the data contained in each attribute (in other words, the width of each label in columns).
space	is the number of horizontal spaces between labels.
C	specifies that null attributes should not be printed. Otherwise, null attributes appear as all blanks. This parameter is optional.

The *size* parameter cannot exceed the page width (80 characters for terminals, and 80/132 characters for printers). Calculate label width as follows:

(*count* * (*size* + *space*) + *indent*) <= (*current page width*)

where *current page width* is the value defined by the TERM or SET-TERM verbs for the terminal or printer.

If a value other than zero is specified for the *indent* parameter, the system prompts you to define headers for each row in a label:

?

The number of ? prompts corresponds to the value entered earlier for *rows*. At each ? prompt, enter the desired header. To avoid defining a header, simply press the RETURN key. Defined headers will appear at the left margin in the *indent* area of the listing.

SREFORMAT: Restructures and sorts items.

The SREFORMAT verb restructures and sorts file items and sends output to a file or to magnetic tape.

SREFORMAT [*file-modifiers*] *filename* [*items*] [*selection*] [*sort*] [*output*] [*modifiers*] [(*options*)]

file-modifiers	can be DICT or ONLY. DICT specifies the file dictionary. ONLY suppresses the default output specification and displays item IDs only.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be transferred. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort by and whether to sort items in ascending or descending order. See the SORT verb for a complete list of the modifiers used to define sort expressions.
output	is the list of attribute-names whose values are to be transferred. If you do not include this parameter, SREFORMAT transfers only item IDs. You can also specify a phrase.

Print limiting output specifications specify which values from multivalued attributes are to be transferred. Use relational operators and values immediately following the name of the multivalued attribute.

modifiers include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

The contents of any attribute in the original file can be made the item IDs for the items in the restructured destination file. This is useful for creating a new file that consists of a subset of the attributes in an existing file. The values that are to be made into item IDs must be unique, however; a null or any other set of identical values will become the item ID of one item, and the data in those items will be stored as multivalues in the item in the destination file.

After SREFORMAT is entered with accompanying parameters, the system displays the following prompt:

```
FILE NAME:
```

At this point you can perform one of three different operations:

1. Transfer items to magnetic tape. To do so, enter the word TAPE at the prompt. You must have already attached the tape unit and set the tape record length with T-ATT. Each item is written to tape as a separate physical tape record (or block). If an item is larger than the tape's block size, the next tape record is used.

2. Transfer items to another file. To do so, enter the name of an existing file as follows:

   [**DICT**] *filename*

   The system copies the selected or specified items to the destination filename.

   If you specify attributes for output and include the ID-SUPP modifier in the ACCESS query, the values in the first attribute of the output specification become the item IDs for the items copied to the destination file. The second attribute listed in the output specification becomes Attribute 1 in the destination file, the third attribute becomes Attribute 2, etc. The destination file can thus consist of a subset of the attributes defined for the original file.

   If you restructure files in this way and you want to generate reports from the destination file, you could copy the Attribute Definition items from the dictionary of the original file to the dictionary of the destination file and then edit the attribute numbers to reflect the new item structure.

3. Transfer items within the same file. To do so, press the RETURN key without entering a response.

   ☞ Use SREFORMAT with care! Most ACCESS verbs generate reports without altering the actual data stored in a database. SREFORMAT, however, is an exception, since it can create new file items and alter existing file items.

*SREPT: Lists multiple items on forms in sorted order.

Not included in the SMA standards. SREPT is a forms generation verb and is not available on all systems. The SREPT verb, like the SFORMS verb, prints file items on such forms as invoices, checks, and order forms in sorted order. Unlike the FORMS and SFORMS verbs, which list one item per page, SREPT (and REPT) can list multiple items as *subpages* on a single page. Using SREPT allows you to explicitly position data either on

the terminal or on a printer page according to *x*- (column) and *y*- (row) coordinates.

Ultimate systems do not have the REPT and SREPT verbs. Instead, subpages are implemented with the M option used with LIST or SORT in conjunction with a forms generation expression. See the FORMS and SFORMS verbs, and Chapter 7, "Forms Generation," for more information about the M option.

SREPT [**DICT**] *filename* [*items*] [*selection*] [*sort*] *output* [modifiers] [(options)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be listed on a form. For a complete description of selection expression syntax, see the LIST verb.
sort	specifies which attributes to sort and whether to sort them in descending or ascending order. See the SORT verb for a description of the four sort modifiers.
output	is the list of attributes to be output on the form. Each attribute specified to be output must be associated with a print code. All forms generation statements must contain at least one print code. Print codes are described in the section "Print Codes." Print limiting output specifications specify which values from multivalued attributes are to be included in the report. Use relational operators and values immediately following the name of the multivalued attribute. Enclose values in double quotes or backslashes.

modifiers include one or more keywords that modify the appearance of the form. These parameters affect headings, footings, and more. For complete information about modifiers, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

The following modifiers behave somewhat differently when used with forms generation verbs: BREAK-ON, FOOTING, HDR-SUPP, HEADING, and ID-SUPP. These modifiers are described in the "Forms Generation Modifiers" section of the REPT verb.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

In addition to the standard ACCESS options, the following options are also available:

A allows you to check printer alignment.

B prints a prestored "background" form along with the data specified in the SFORMS statement.

Z on multipage forms, resets footing page number to one.

These nonstandard options are fully described in the "Forms Generation Options" section of the REPT verb.

After SREPT is entered, the system prompts for subpage size:

```
Subpage size>
```

Enter the number of lines each subpage should contain. For example, to define 6 subpages on each page of 62 lines (2 lines are reserved for the heading), enter 10. Each subpage extends the full width of the page.

The system does not split items; if an entire subpage will not fit at the bottom of a page, the system prints the subpage on the next page.

SREPT applies the format specified in the command line to the *output* specifications. SREPT prints one item per form; a form can be either single-page or multipage. Data can be printed anywhere on a page, and can be printed on just one page or on every page of a form. In addition to specifying how the data is to be printed on a form, SREPT can be used to specify the layout of the form itself.

The @W and @D print codes (described in the next section, "Print Codes") can be used to define vertical windows in which the data for multivalued attributes can be output. If the data for a multivalued attribute does not fit in a window, the remaining data is printed on the next page. A form can contain up to six separate windows.

Ultimate systems use a modified version of the FORMS verb syntax. The FORMS verb itself is not supported by Ultimate; instead, a *forms generation expression* can be included in the LIST verb syntax. Forms generation expressions have essentially the same syntax as they do in the FORMS verb, except that the verb used is LIST instead of FORMS. Other differences in usage are noted in the following pages.

As was mentioned earlier, the FORMS and SFORMS verbs print one item per page in sorted order. REPT and SREPT can be used to print more than one item on a page.

Print Codes

Each attribute name included on a form must be associated with a print code as follows:

@code **(** *x,y* [**,***z*] **)** **:** *attribute-name* [**1,***n*]

To define a character string that prints in a specified location on the form, use the following syntax:

@*code* (*x*,*y* [,*z*]) : "*string*" [1,*n*]

code	is the print code associated with the attribute. The available print codes are shown in Table A-4.
x	is the horizontal position (column) on the page where the data begins. The leftmost position is column 0.
y	is the vertical position (row) on the page where the data begins. The top row is row 0, which is reserved for the heading.
z	is extra data required by the @D and @W print codes.
attribute-name	is the name of the attribute whose data is printed at the specified position.
string	is a character string that is printed at the specified position.
n	prints only the first *n* characters of the data for this attribute.

Table A-4 summarizes the available print codes.

Table A-4. Print Codes.

Print Code	Description
@[A](*x*,*y*)	Prints the data for an attribute on every page of a multipage form. The "A" is optional.
@C(*x*,*y*)	Creates an audit trail for a series of forms. The @C code can also be used to serialize the forms. To suppress serialization numbers on the form (but not in the audit file), specify (–1) in place of the *x* and *y* coordinates. (You *must* use either the *x*-*y* coordinates or –1 with the @C code.)
@D(*x*,*y*,*z*)	Not available on all systems. The @D print code prints data for multivalued attributes in double-depth windows. This makes it possible to define two lines

	of output at a time. *z* specifies the bottommost row of a window whose top-most row is defined by *y*. See also the WINDOW keyword. You must add an S to the end of the *x* parameter for an attribute to appear on every second output line.
@F(*x*,*y*)	Prints the data for an attribute on only the first page of a multipage form.
@L(*x*,*y*)	Prints the data for an attribute on only the last page of a multipage form.
@M(*x*,*y*,"*text*")	Prints the specified text on all but the last page of a multipage form. The data for the attribute prints on the last page of the form. On a single-page form, the data for the attribute is printed.
@W(*x*,*y*,*z*)	Not available on all systems. The @W print code prints data for multivalued attributes in windows. *x* specifies the columns where attribute values are to start being printed. *z* specifies the bottom-most row of a window whose topmost row is defined by *y*. See also the WINDOW keyword.

For complete information about using print codes, see the REPT verb and Chapter 7, "Forms Generation."

SSELECT: Selects and sorts items for further processing.

The SSELECT verb creates a temporary select-list of sorted file items to be processed by the next TCL or ACCESS statement, or by other processors such as the Editor, PICK/BASIC, or PROC. Creating a select-list is a useful way to define and operate on a subset of items in a database.

SSELECT [**DICT**] *filename* [*items*] [*selection*] [*sort*] [*output*]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is a list of individual item IDs or an item selection expression. Enclose each item ID in single quotes. If you do not specify *items*, SSELECT selects all items in *filename*.

selection specifies one or more conditions that data in an item must meet to be included in the select-list. For a complete description of selection expression syntax, see the LIST verb.

sort specify which attributes to sort by and whether to sort items in ascending or descending order. See the SORT verb for a complete list of the modifiers used to define sort expressions.

output is the name of one or more attributes whose values are to be selected. Each value becomes a separate element in the select-list. If output is specified, no item IDs will be selected.

SSELECT creates a temporary select-list containing sorted item IDs or data values of items specified by *items* or the selection expression. The items referenced by the select-list will be processed by the next verb you execute. For instance, before invoking the Editor you might create a select-list as a way of specifying which items you want to edit.

☞ Only the statement immediately following the SSELECT statement will have access to the select-list. In other words, you must use the select-list immediately, or you lose it!

To permanently save the select-list, use the SAVE-LIST verb. Once a select-list is saved, you can retrieve it at any time with the GET-LIST verb.

A select-list can reference data in any file, not just the file specified in the original SSELECT statement. If two files have similar items with the same item IDs, you can create a select-list from one file, then use it to operate on items from the other file.

STAT: Lists statistics for a specified attribute.

The STAT verb totals the numeric elements contained in a specified attribute of a file. It also counts the number of items selected and averages the data contained in the attribute.

STAT [DICT] *filename* [*items*] [*selection*] [*attribute-name*] [*modifiers*] [**(** *options* **)**]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the total. For a complete description of selection expression syntax, see the LIST verb.
attribute-name	is the attribute that you want to total. If you do not specify an attribute, the item IDs are totalled.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

The output of the STAT command is displayed in the following format:

STATISTICS OF *attribute*
TOTAL = *tot* AVERAGE = *avg* COUNT = *ct*
WHERE

attribute	is the name of the attribute for which the statistics are produced.

tot	is the total of the data values in the attribute.
avg	is the total of the data values divided by the count.
ct	is the number of items selected.

SUM: Totals the data elements in a numeric attribute.

The SUM verb totals the data elements in a numeric attribute.

SUM [**DICT**] *filename* [*items*] [*selection*] [*attribute-name*] [*modifiers*] [**(** *options* **)**]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be included in the sum. For a complete description of selection expression syntax, see the LIST verb.
attribute-name	is the attribute that you want to sum. If you do not specify an attribute, the item IDs are summed.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as

commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

-DUMP: Copies items to magnetic tape.

The T-DUMP verb transfers a copy of all or selected file items in random order to magnetic tape. This verb also creates a tape label and writes an End-Of-File (EOF) mark on the tape after the transfer is complete.

T-DUMP [**DICT**] *filename* [*items*] [*selection*] [**HEADING** "*text*"] [*modifiers*] [(*options*)]

DICT — specifies the file dictionary. When copying dictionary items, T-DUMP does not copy any File Definition items to tape.

filename — is the name of the file.

items — is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.

selection — specifies one or more conditions that data in an item must meet to be copied. For a complete description of selection expression syntax, see the LIST verb.

text — is added to the standard tape label via the HEADING modifier.

modifiers — include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."

The following modifiers operate differently with T-DUMP:

HDR-SUPP suppresses the creation of a tape label.

ID-SUPP suppresses listing of item IDs during the copy-to-tape operation.

options include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

Before using T-DUMP, use T-ATT to attach the tape unit and set the tape record length. You can select a block length, or you can accept the system default block length of 4000 bytes for 1/2-inch tape or 16,896 bytes for 1/4-inch cartridge tape. In addition, the logical tape unit should already have been selected with the T-SELECT verb.

If the tape's write protection is on or if no write ring is present, the following message is displayed:

```
WRITE PROTECTED
CONTINUE/QUIT (C/Q)?
```

Remove the write protection or install a write ring, then enter "C" to continue, or enter "Q" to quit.

T-DUMP automatically writes a label to the tape in the following format:

[**DICT**] *filename*

You can add text to this label with the *text* parameter, or you can suppress the tape label completely by including the HDR-SUPP keyword in the T-DUMP command line.

For information about reading a tape created with T-DUMP, see the T-LOAD verb.

T-LOAD: Restores items from magnetic tape to disk.

The T-LOAD verb restores file items that were previously copied to tape with either the T-DUMP or S-DUMP verbs. These file items can be copied only to an existing file.

T-LOAD [**DICT**] *filename* [*items*] [*selection*] [*modifiers*]
[(*options*)]

DICT	specifies the file dictionary.
filename	is the name of the file.
items	is the list of individual item IDs or an item selection expression. Enclose each item ID in single quotes.
selection	specifies one or more conditions that data in an item must meet to be restored. For a complete description of selection expression syntax, see the LIST verb.
modifiers	include one or more keywords that specify the report format. These parameters affect headers, footers, spacing, totalling column figures, control breaks, and more. For complete information about using these keywords, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords."
options	include one or more single-character codes that specify the report format and direct or modify output. They must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or any delimiters such as commas. For complete information about using these parenthetical options, see Appendix B, "ACCESS Keywords."

Before using T-LOAD, attach the tape unit and set the tape record length by entering T-ATT. You can select a block length, or you can accept the system default block length of 4000 bytes for 1/2-inch tape or 16,896

bytes for 1/4-inch cartridge tape. In addition, the logical tape unit should already have been selected with the T-SELECT verb.

When the restore operation is complete, the tape is positioned at the End-Of-File (EOF) mark.

T-LOAD will not restore any items from tape that already exist in the file. Use the O option if you want to overwrite existing items.

You can retrieve and list data from tape without restoring it by using the TAPE modifier with ACCESS verbs such as LIST and LIST-ITEM. For more information about the TAPE modifier, see Chapter 4, "Formatting Reports," and Appendix B, "ACCESS Keywords.".

For information about copying items to magnetic tape, see the T-DUMP and S-DUMP verbs.

APPENDIX B

ACCESS Keywords

Appendix B contains an alphabetic quick reference of all ACCESS keywords, with examples illustrating how to use them. Also included are tables listing the keywords according to their different categories: connectives, relational operators, modifiers, and options. Keywords that are not included in the SMA standards are marked with an asterisk.

Table 1: Connectives.

Selection connectives select items that meet certain conditions:

***LIKE**	Select items that "sound like" a specified constant. Not implemented on all systems. (On Prime INFORMATION and uniVerse systems, LIKE is synonymous with MATCHING.)
***MATCHING**	Select items whose data matches a specified constant. Not implemented on all systems.
***SAID**	Select items that "sound like" a specified constant. Not implemented on all systems.
***SPOKEN**	Select items that "sound like" a specified constant. Not implemented on all systems.
USING	Use a different file as dictionary.
***WITHIN**	List subitems.

Logical connectives select items that meet multiple conditions:

AND, &	All conditions must be met.
OR, !	One or more conditions must be met (default).

Throwaway connectives make an ACCESS sentence sound more like an English-language sentence:

A	**DATA**	**ITEMS**
AN	**FILE**	**OF**
ANY	**FOR**	**OR**
ARE	**IN**	**THE**

Table 2: Relational Operators.

Relational operators compare an attribute with a constant or with another attribute:

EQ, =	Equal to (default)
NE, #	Not equal to
LT, <, BEFORE	Less than
GT, >, AFTER	Greater than
LE, <=, =<	Less than or equal to
GE, >=, =>	Greater than or equal to
NO, NOT	Null

Table 3: Modifiers.

File modifiers modify the filename and must precede it in the ACCESS query.

DICT	Select items from the file dictionary.
ONLY	List item IDs only.

Selection expression modifiers specify the conditions that must be met in order for items to be selected:

EACH	All values in a multivalued attribute must meet criteria for the item to be selected.
EVERY	All values in a multivalued attribute must meet criteria for the item to be selected.
IF	Values in an attribute must meet specified criteria for the item to be selected.
TAPE	Select items from a tape.
WITH	Values in an attribute must meet specified criteria for the item to be selected.
WITH NO	If values in an attribute do not meet specified criteria, the item is selected.
WITHOUT	If values in an attribute do not meet specified criteria, the item is selected.

Sort expression modifiers specify the sort order in SORT statements:

BY	Sort in ascending order.
BY-DSND	Sort in descending order.
BY-EXP	Sort multivalues in ascending order.
BY-EXP-DSND	Sort multivalues in descending order.

Output modifiers modify the ACCESS report:

BREAK-ON	Break output when a specified attribute's value changes.
CAPTION	Print text for total line
COL-HDR-SUPP	Suppress default column headings, page headings, and end-of-list message.
DBL-SPC	Double-space items.
DET-SUPP	Suppress detail lines.
FOOTING	Print a specified footing.
GRAND-TOTAL	Print text for Total line.
HEADING	Print a specified heading.
HDR-SUPP	Suppress the page heading and end-of-list message.
ID-SUPP	Suppress item IDs.
LPTR	Direct output to printer.
NOPAGE	Suppress pagination for terminal output.
SUPP	Suppress the page heading and end-of-list message.
TOTAL	Sum values for a specified attribute.

Table 4: Options.

Options must be enclosed in parentheses, can be entered in any order, and need not be separated by spaces or delimiters:

B	Suppress initial terminal line feed before output.
C	Suppress default column headings, page headings, and end-of-list message.
D	Suppress detail lines. With COPY-LIST, delete original select-list.
***F**	Generate form feed.
H	Suppress page heading and end-of-list message.
I	Suppress item IDs.
N	Suppress pagination for terminal output.
***O**	Overwrite item (t-load).
P	Direct output to printer.
***T**	Direct output to terminal (default).
***Y**	Print translation of A-correlative DICT entries.

=

Synonym for "Equal To." See EQ for description.

#

Synonym for "Not Equal To." See NE for description.

>

Synonym for "Greater Than." See GT for description.

>=

Synonym for "Greater Than or Equal To." See GE for description.

<

Synonym for "Less Than." See LT for description.

<=

Synonym for "Less Than or Equal To." See LE for description.

A/AN

A and AN are throwaway connectives that can be used in an ACCESS query to make it sound more like an English-language sentence. They have no effect.

Example:

The following two statements produce the same report:

```
>SORT ORDERS WITH A TOTAL.AMT > "100"
>SORT ORDERS WITH TOTAL.AMT > "100"
```

The A connective is ignored in the first statement.

AFTER

Synonym for "Greater Than." See GT for description.

AND

AND is a logical connective used to specify multiple selection expressions. AND specifies that all conditions must be met for an item to be selected.

AND takes precedence over the OR connective when both appear in the same query. A maximum of nine AND clauses can be included in a single ACCESS query.

OR is assumed when a logical connective is not explicitly entered between two selection expressions.

Example:

The following example lists all McCoys who live in Berkeley:

```
>LIST CUSTOMERS WITH LAST-NAME = "MCCOY" AND
  CITY = "BERKELEY" LAST-NAME FIRST-NAME
```

ANY

ANY is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>LIST ANY ORDERS WITH TOTAL.AMT < "100" AND > "25"
>LIST ORDERS WITH TOTAL.AMT < "100" AND > "25"
```

The ANY connective is ignored in the first statement.

ARE

ARE is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>LIST CUSTOMERS IF ORDERS ARE > "200"
>LIST CUSTOMERS IF ORDERS > "200"
```

The ARE connective is ignored in the first statement.

BEFORE

Synonym for "Less Than." See LT for description.

BREAK-ON

BREAK-ON is an output modifier used with LIST and SORT verbs that specifies which attribute is to be used to create breaks within a report. A control break (indicated by three asterisks or by user-specified text) occurs when the data for the specified attribute changes.

The BREAK-ON modifier is often used with the TOTAL modifier to calculate subtotals for numeric data. Normally the data is sorted by the

same attribute as the one specified by BREAK-ON so that the same set of data is processed and displayed together.

To set control breaks for a multivalued attribute, use the BY-EXP or BY-EXP-DSND modifiers to display individual detail lines for each value.

The syntax for the BREAK-ON modifier is:

BREAK-ON *attribute* [" [*text*] ['*options* '] [*text*] "]

attribute	is the attribute whose changing data causes a break. ACCESS evaluates the first 24 characters of the data, from left to right. The break is displayed in this attribute's column.
text	is printed on the break line. This text replaces the three asterisks that are output by default. Enclose the text in double quotes.

options affect the processing of the break. Options must be enclosed in single quotes. They include:

B	(Break.) B includes the value of the attribute causing the break in the heading or footing. It must be used in conjunction with the B option of the HEADING or FOOTING modifiers.
D	(Data.) D suppresses printing of the break line if there is only one line of detail for a section, but leaves a blank line between items.
L	(Line.) L suppresses the blank line preceding the break line. The L option has no effect when specified with the U option.
P	(Page.) P begins a new page after every control break.
R	(Rollover.) R forces all data associated with a control break to appear on the same page.
U	(Underline.) U underlines all totals for each control break.

V	(Value.) V inserts the current data for the attribute into the text on the break line.
' '	(Two single quotes.) ' ' inserts one single quote in *text*.

In the ACCESS query, the break attribute should be entered immediately after the BREAK-ON modifier and its accompanying parameters.

The attribute specified by the BREAK-ON modifier is automatically included in the output specifications for the report, so it does not need to be specified separately in the ACCESS query.

To suppress the printing of the BREAK-ON attribute, place a " \ " in Attribute 3 and a "0" in Attribute 10 of the Attribute Definition item (in the file dictionary) that defines the BREAK-ON attribute.

Example:

The following example sorts orders by last name and breaks the display at the occurrence of each new name:

```
>SORT ORDERS BY LAST-NAME BREAK-ON LAST-NAME
  " 'L' " TOTAL.AMT
```

```
PAGE   2                                          15:46:36  01 NOV 1989

ORDERS.... LAST-NAME... TOTAL.AMT.

10113      JENKINS         $357.00
           ***

10101      JOHNSON          $18.00
10107      JOHNSON         $799.00
10111      JOHNSON         $126.00
           ***

10102      LEARY            $72.00
10108      LEARY           $252.00
10112      LEARY           $231.00
10114      LEARY            $47.00
           ***

10106      ORLANDO          $63.00
           ***
```

The L option suppressed the printing of the blank line before the break line.

BY

BY is a modifier that sorts items in ascending order by comparing the ASCII values of data in the specified attribute. The designated attribute is called a sort-key. Multiple BY clauses can be included in an ACCESS query to specify primary and secondary sort-keys. The sort-keys are processed from left to right.

The syntax for the BY modifier is:

BY *attribute* [**BY** *attribute* ...]

If BY is used with a multivalued attribute, only the first value will be sorted. To sort the individual values in a multivalued attribute, use the BY-EXP modifier.

Example:

The following example sorts customers by last name and first name. LAST-NAME is the primary sort-key and FIRST-NAME is the secondary sort-key:

```
>SORT CUSTOMERS BY LAST-NAME BY FIRST-NAME
  LAST-NAME  FIRST-NAME
```

```
PAGE   1                                          10:58:13  01 NOV 1989

CUSTOMERS. Last-Name  First-Name

MASHX5777  ASH        MARY
JBOHA5422  BOHANNON   JOHN
JBUCK6488  BUCKLER    JULIE
DEDGE6635  EDGECOMB   DAVID
HJENK7129  JENKINS    HAROLD
AJOHN5396  JOHNSON    ALICE
HJOHN7265  JOHNSON    HENRY
BLEAR6803  LEARY      BILL
AORLA5993  ORLANDO    AMY
JPEER5993  PEERCE     JAN
SPIRS5289  PIRS       SANDRA

11 ITEMS LISTED.

>
```

The Johnsons were sorted first by last name, then by first name.

BY-DSND

BY-DSND is a modifier that sorts items in descending order. The designated attribute is called a sort-key. Multiple BY-DSND clauses can be included in an ACCESS query to specify primary and secondary sort-keys. The sort-keys are processed from left to right.

The syntax for the BY-DSND modifier is:

BY-DSND *attribute* [**BY-DSND** *attribute* ...]

If BY-DSND is used with a multivalued attribute, only the first value is sorted. To sort all the values in a multivalued attribute, use the BY-EXP-DSND modifier.

Example:

The following example sorts orders in descending order by amount:

```
>SORT ORDERS BY-DSND TOTAL.AMT TOTAL.AMT DATE
```

```
PAGE   1                                          11:14:03  01 NOV 1989

ORDERS.... AMOUNT....     DATE.......

10110         $1088.00    25 DEC 1987
10104          $827.00    27 MAR 1988
10107          $761.00    17 MAR 1988
10113          $340.00    02 JUN 1988
10108          $240.00    30 MAY 1988
10105          $229.00    17 MAR 1988
10112          $220.00    22 MAR 1988
10120          $160.00    04 SEP 1988
10109          $140.00    22 OCT 1987
10111          $120.00    29 SEP 1988
10115          $109.00    11 JUL 1988
10102           $69.00    09 APR 1988
10106           $60.00    16 APR 1988
10121           $60.00    19 MAY 1988
10114           $45.00    13 DEC 1987
10116           $40.00    28 SEP 1989
10119           $40.00    23 JUN 1987
10122           $40.00    19 OCT 1987
10123           $40.00    01 AUG 1989
10124           $40.00    24 JUN 1987
```

BY-EXP

BY-EXP is a modifier that first separates values in a multivalued attribute and then sorts them in ascending order. EXP stands for exploded, meaning that each value in the multivalued attribute is sorted separately.

The syntax for the BY-EXP modifier is:

BY-EXP *attribute* [**BY-EXP** *attribute* ...]

Example:

The following example sorts orders by last name, then sorts the multivalued item TITLE. The BREAK-ON modifier separates the report by LAST-NAME:

```
>SORT ORDERS BY LAST-NAME BY-EXP TITLE
  BREAK-ON LAST-NAME TITLE
```

```
PAGE   1                                          13:56:41  01 NOV 1989

ORDERS.... LAST-NAME... TITLE.............................

10122      BUCKLER      WRITING COMMERCIAL APPLICATIONS

           ***

10105      EDGECOMB     DATABASE MANAGEMENT SYSTEMS
10115      EDGECOMB     DATABASE MANAGEMENT SYSTEMS
10115      EDGECOMB     OPERATING SYSTEM CONCEPTS
10105      EDGECOMB     WORD PROCESSING
10105      EDGECOMB     WRITING COMMERCIAL APPLICATIONS

           ***

10113      JENKINS      OPERATING SYSTEM CONCEPTS
10113      JENKINS      WORD PROCESSING

           ***

10101      JOHNSON      DATABASE MANAGEMENT SYSTEMS
10107      JOHNSON      DATABASE MANAGEMENT SYSTEMS
```

If the BY modifier had been used in place of BY-EXP, TITLE would have been sorted by the first title of each order only. The remaining titles would not have been sorted.

BY-EXP-DSND

BY-EXP-DSND is a modifier that first separates the values in a multivalued attribute and then sorts them in descending order. EXP stands for exploded, meaning that each value in the multivalued attribute is sorted separately.

The syntax for the BY-EXP-DSND modifier is:

BY-EXP-DSND *attribute* [**BY-EXP-DSND** *attribute* ...]

Example:

The following example sorts orders by last name, then sorts the multivalued item PRICE in descending order. The BREAK-ON modifier separates the report by LAST-NAME:

```
>SORT ORDERS BY LAST-NAME BY-EXP-DSND PRICE
  BREAK-ON LAST-NAME SHORT.TITLE
```

```
PAGE   2                                          14:17:26  01 NOV 1989

ORDERS.... LAST-NAME... SHORT TITLE........................ PRICE...

10122      BUCKLER      WRITING                                $24.50

           ***

10105      EDGECOMB     WRITING                                $24.50
10105      EDGECOMB     WORD                                   $22.98
10115      EDGECOMB     OPERATING                              $18.75
10105      EDGECOMB     DATABASE                                $9.95
10115      EDGECOMB     DATABASE                                $9.95

           ***

10113      JENKINS      WORD                                   $22.98
10113      JENKINS      OPERATING                              $18.75

           ***

10107      JOHNSON      OPERATING                              $18.75
10111      JOHNSON      OPERATING                              $18.75
```

If the BY-DSND modifier had been used in place of BY-EXP-DSND, PRICE would have been sorted by the price of the first title of each order only. The remaining titles would not have been sorted.

C

C is an option that suppresses the printing of the default page heading (page, time, and date), column headings, and end-of-list message (COL-HDR-SUPP).

Example:

The following example suppresses the printing of the default headings:

```
>SORT CUSTOMERS BY STATE STATE (C)

AORLA5993  CA
JPEER5993  CA
DEDGE6635  FL
HJENK7129  IN
JBUCK6488  IN
MASHX5777  IN
AJOHN5396  KY
BLEAR6803  MA
JBOHA5422  MA
HJOHN7265  NB
SPIRS5289  NC

>
```

COL-HDR-SUPP

COL-HDR-SUPP is an output modifier that suppresses the printing of the default page heading (page, time and date), column headings, and end-of-list message.

The C option can be used in place of COL-HDR-SUPP.

COL-HDR-SUPP

Example:

The following example suppresses the printing of the default headings:

```
>SORT CUSTOMERS BY STATE STATE COL-HDR-SUPP
```

```
AORLA5993  CA
JPEER5993  CA
DEDGE6635  FL
HJENK7129  IN
JBUCK6488  IN
MASHX5777  IN
AJOHN5396  KY
BLEAR6803  MA
JBOHA5422  MA
HJOHN7265  NB
SPIRS5289  NC

>
```

D

D is an option that suppresses detail lines in ACCESS statements (DET-SUPP).

☞ When the D option is used with the COPY-LIST verb, it deletes the source select-list after copying it.

DATA

DATA is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>LIST DATA ORDERS
>LIST ORDERS
```

The DATA connective is ignored in the first statement.

DBL-SPC

DBL-SPC is an output modifier that double-spaces between items in a report. Values in multivalued attributes are single-spaced.

Example:

The following example generates a report with a space between each item:

```
>SORT ORDERS BY DATE WITH DATE < "12/31/87" TITLE
 DATE DBL-SPC
```

```
PAGE   1                                          11:46:26  01 NOV 1989

ORDERS.... TITLE.............................     DATE.......

10119      OPERATING SYSTEM CONCEPTS               23 JUN 1987

10124      OPERATING SYSTEM CONCEPTS               24 JUN 1987

10122      WRITING COMMERCIAL APPLICATIONS         19 OCT 1987

10109      WORD PROCESSING                         22 OCT 1987

10114      DATABASE MANAGEMENT SYSTEMS             13 DEC 1987

10110      OPERATING SYSTEM CONCEPTS               25 DEC 1987
           WRITING COMMERCIAL APPLICATIONS
           WORD PROCESSING
           DATABASE MANAGEMENT SYSTEMS


6 ITEMS LISTED.

>
```

DET-SUPP

DET-SUPP is an output modifier that suppresses detail lines when it is used in a query that contains the BREAK-ON or TOTAL modifiers. Only break lines are displayed.

The D option can be used in place of DET-SUPP.

Example:

The following example suppresses detail lines:

```
>SORT ORDERS BY-EXP TITLE BREAK-ON TITLE TOTAL
  TOTAL.AMT DET-SUPP
```

```
PAGE   1                                        11:49:22  01 NOV 1989

ORDERS.... TITLE.............................. AMOUNT....

           DATABASE MANAGEMENT SYSTEMS           $3146.00

           OPERATING SYSTEM CONCEPTS             $3914.00

           WORD PROCESSING                       $2724.00

           WRITING COMMERCIAL APPLICATIONS       $2844.00
***                                             $12628.00

35 ITEMS LISTED.

>
```

All orders are sorted by title and the amount is then totalled for each title, but the report only prints the name of each title once, along with the total amount. Detail lines for each title ordered are not printed.

DICT

DICT is a file modifier that generates a report from the file dictionary rather than the data file. DICT must be specified immediately before the filename in an ACCESS query.

Example:

The following example lists all Attribute Definition items with an attribute number of 3:

```
>LIST DICT CUSTOMERS WITH A/AMC = "3"
```

EACH

EACH is a selection modifier that selects an item only if every value in a multivalued attribute meets the specified condition. EACH must be specified immediately after the WITH modifier.

Synonym: EVERY

Example:

The following example lists only those items for which all values in the multivalued attribute LINE.AMT are greater than or equal to 100:

```
>LIST ORDERS WITH EACH LINE.AMT GE "100"
```

*END-WINDOW

Not implemented on all systems. Ultimate uses the END-WINDOW modifier to end a phrase that defines a window for printing multivalues on forms. WINDOW phrases are used in conjunction with forms generation expressions. A WINDOW phrase must always be followed by the END-WINDOW modifier. See the WINDOW modifier for more information.

EQ

EQ is the relational operator EQUAL TO, used in selection expressions to compare data in an attribute to a constant.

Synonyms: EQUAL, =

EQ

Example:

The following example lists all customers whose last name is Johnson:

```
>LIST CUSTOMERS WITH LAST-NAME EQ "JOHNSON"
```

EQUAL

Synonym for "Equal To." See EQ for description.

EVERY

Synonym for EACH. See EACH for description.

*F

F is an option that generates a form-feed for each item. Used only with the LIST-ITEM, and SORT-ITEM verbs.

FILE

FILE is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>SORT THE CUSTOMERS FILE BY LAST-NAME
>SORT CUSTOMERS BY LAST-NAME
```

Both the connectives THE and FILE are ignored in the first statement.

FOOTING

FOOTING is an output modifier that defines a footing for the bottom of each page of a report.

The syntax for the FOOTING modifier is:

FOOTING "[*text*] ['*options*'] [*text*] ['*options*']..."

text is text that appears in the footing.

options can be interspersed anywhere within the text in order to modify those parts of the text. They must be enclosed in single quotes. *options* include the following:

B [*n*]	(Break.) B inserts the value of the attribute causing a control break. It must be used in conjunction with the B option of the BREAK-ON modifier. *n* inserts the control-break value left-justified in a field of *n* blanks.
C	(Center.) C centers text.
D	(Date.) D inserts the current date in the format *dd mmm yyyy*.
F [*n*]	(File.) F inserts the filename. *n* inserts the filename left-justified in a field of *n* blanks.
I [*n*]	(Item.) I inserts the item ID. *n* inserts the item ID left-justified in a field of *n* blanks.
L	(Line-feed.) L inserts a carriage return and a line-feed.
P [*n*]	(Page.) P inserts the page number, right-justified in a field of four blanks. *n* inserts the page number left-justified in a field of *n* blanks.
PN	PN inserts the page number, left-justified.
T	(Time.) T inserts the current time and date in the format *hh:mm:ss dd mmm yyyy*. The time appears in 24-hour format.
' '	(Two single quotes.) ' ' prints a single quote in the footing text.

Text and options can appear in any order. FOOTING displays the information in the order specified. Spaces can be used anywhere in the footing text to make it more legible. The entire string of text and options must be enclosed in double quotes.

Example:

The following example produces a report from the BOOKS file that includes a four-line footing:

```
>SORT BOOKS BY PRICE TITLE PRICE HDR-SUPP
  ID-SUPP FOOTING "Current Titles and Prices'CL'====
  ==========================================
  ==============='CLL'PAGE:'P'
  FILE: 'F'    ITEM: 'I'"
```

```
TITLE.............................. PRICE...

DATABASE MANAGEMENT SYSTEMS            $9.00
OPERATING SYSTEM CONCEPTS             $20.00
WRITING COMMERCIAL APPLICATIONS       $20.00
WORD PROCESSING                       $20.00









                    Current Titles and Prices
  ============================================================

PAGE: 1                          FILE: BOOKS          ITEM: QR02
```

The C option centers text, and the L option inserts a line feed. The P option prints the page number, the F option prints the current filename, and the I option prints the current item ID.

OR

FOR is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>LIST CUSTOMERS FOR STATE
>LIST CUSTOMERS STATE
```

The FOR connective is ignored in the first statement.

GE

GE is the relational operator GREATER THAN OR EQUAL TO, used in selection expressions to compare data in an attribute to a constant.

Synonym: >=

Example:

The following example sorts orders with amounts greater than or equal to "100":

```
>SORT ORDERS WITH TOTAL.AMT GE "100" TOTAL.AMT
```

GRAND-TOTAL

GRAND-TOTAL is an output modifier that specifies text to be printed on the Grand Total line of a report. The specified text is printed left-justified in the first column of the report, replacing the three asterisks that normally appear.

☞ Text entered for the Grand Total should not exceed the character-width specification for the first column; it could overwrite the total figures.

The syntax for the GRAND-TOTAL modifier is:

GRAND-TOTAL " [*text*] ['*options*'] [*text*] "

text is the text to be printed on the Grand Total line of the report.

options must be enclosed in single quotes. They include the following:

L	(Line.) L suppresses the blank line that precedes the Grand Total line. This option is ignored if the U option is used in the same query.
P	(Page.) P prints Grand Totals on a separate page.
U	(Underline.) U underlines all totalled attributes with a row of equal signs.

The entire string of text and options must be enclosed in double quotes.

Example:

The following report includes a Grand Total line that reads "TOTAL SALES":

```
>SORT ORDERS BY-EXP TITLE BREAK-ON TITLE TOTAL
  TOTAL.AMT GRAND-TOTAL "TOTAL SALES:'U' "
```

```
PAGE   3                                        15:35:18  01 NOV 1989

ORDERS. TITLE............................. AMOUNT....

10108   WRITING COMMERCIAL APPLICATIONS        $236.50
10110   WRITING COMMERCIAL APPLICATIONS       $1301.13
10112   WRITING COMMERCIAL APPLICATIONS        $229.25
10120   WRITING COMMERCIAL APPLICATIONS        $196.00
10122   WRITING COMMERCIAL APPLICATIONS         $49.00
10123   WRITING COMMERCIAL APPLICATIONS         $49.00

        ***                                   $3259.11
                                             ==========
TOTAL SALES:                                 $14084.67

35 ITEMS LISTED.

>
```

The U option underlines the totalled attribute.

GT

GT is the relational operator GREATER THAN, used in selection expressions to compare data in an attribute to a constant.

Synonyms: >, AFTER

Example:

The following example sorts orders with amounts greater than "100".

```
>SORT ORDERS BY TOTAL.AMT WITH TOTAL.AMT GT
  "100" TOTAL.AMT
```

H

H is an option that suppresses the printing of the default page heading (page, time, date) and the end-of-list message (HDR-SUPP).

Example:

The following example sorts customers by state and suppresses the default page heading:

```
>SORT CUSTOMERS BY STATE STATE (H)
```

```
CUSTOMERS. STATE

AORLA5993     CA
JPEER5993     CA
DEDGE6635     FL
HJENK7129     IN
JBUCK6488     IN
MASHX5777     IN
AJOHN5396     KY
BLEAR6803     MA
JBOHA5422     MA
HJOHN7265     NB
SPIRS5289     NC

>
```

HDR-SUPP

HDR-SUPP is an output modifier that suppresses the printing of the default page heading (page, time, date) and the end-of-list message. When used with the T-DUMP, S-DUMP, or T-LOAD verbs, HDR-SUPP suppresses display of the tape label.

The H option can be used in place of HDR-SUPP.

Synonym: SUPP

Example:

The following example sorts customers by state and suppresses the default page heading:

```
>SORT CUSTOMERS BY STATE STATE HDR-SUPP
```

```
CUSTOMERS. STATE

AORLA5993      CA
JPEER5993      CA
DEDGE6635      FL
HJENK7129      IN
JBUCK6488      IN
MASHX5777      IN
AJOHN5396      KY
BLEAR6803      MA
JBOHA5422      MA
HJOHN7265      NB
SPIRS5289      NC

>
```

HEADING

HEADING is an output modifier that defines a heading to replace the default heading for the top of each page of a report.

The syntax for the HEADING modifier is:

HEADING "[*text*] ['*options*'] [*text*] ['*options*']..."

text is text that appears in the heading.

options can be interspersed anywhere within the text to modify those parts of the text. They must be enclosed in single quotes. *options* include the following:

B [*n*]	(Break.) B inserts the value of the attribute causing a control break. It must be used in conjunction with the B option of the BREAK-ON modifier. *n* inserts the control-break value left-justified in a field of *n* blanks.
C	(Center.) C centers text.
D	(Date.) D inserts the current date in the format *dd mmm yyyy*.
F [*n*]	(File.) F inserts the filename. *n* inserts the filename left-justified in a field of *n* blanks.
I [*n*]	(Item.) I inserts the item ID. *n* inserts the item ID left-justified in a field of *n* blanks.
L	(Line-feed.) L inserts a carriage return and a line-feed.
P [*n*]	(Page.) P inserts the page number, right-justified in a field of four blanks. *n* inserts the page number left-justified in a field of *n* blanks.
PN	PN inserts the page number, left-justified.
T	(Time.) T inserts the current time and date in the format *hh:mm:ss dd mmm yyyy*. The time appears in 24-hour format.
' '	(Two single quotes.) ' ' prints a single quote in the heading text.

Text and options can appear in any order. HEADING displays the information in the order specified. Spaces can be used anywhere in the

heading text to make it more legible. The entire string of text and options must be enclosed in double quotes.

Example:

The following example produces a report from the BOOKS file that includes a four-line heading:

```
>SORT BOOKS BY PRICE TITLE PRICE ID-SUPP HEADING
  "Current Titles and Prices'CL'=============
  ============================================
  =============='CL'PAGE: 'P'                    FILE:
  'F'     ITEM: 'IL' "
```

```
                        Current Titles and Prices
================================================================================
PAGE:    1                              FILE: BOOK-CATALOG     ITEM: N01

TITLE............................. PRICE...

DATABASE MANAGEMENT SYSTEMS             $9.95
OPERATING SYSTEM CONCEPTS              $18.75
WRITING COMMERCIAL APPLICATIONS        $22.98
WORD PROCESSING                        $24.50

>
```

The C option centers text, and the L option inserts a line feed. The P option prints the page number, the F option prints the current filename, and the I option prints the current item ID.

I is an option that suppresses printing of item IDs in a report (ID-SUPP).

Example:

The following example suppresses the printing of item IDs in a BOOKS report:

```
>SORT BOOKS BY TITLE TITLE PRICE (I)
```

```
PAGE   1                                          12:16:05  01 NOV 1989

TITLE............................ PRICE...

DATABASE MANAGEMENT SYSTEMS            $9.95
OPERATING SYSTEM CONCEPTS             $18.75
WORD PROCESSING                       $22.98
WRITING COMMERCIAL APPLICATIONS       $24.50

4 ITEMS LISTED.

>
```

ID-SUPP

ID-SUPP is an output modifier that suppresses the display of item IDs in a report. Without this keyword, item IDs automatically appear as the first column in a report in addition to the attributes explicitly specified in the ACCESS query.

When used with the T-DUMP, S-DUMP, OR T-LOAD verbs, ID-SUPP suppresses the listing of item IDs copied to or retrieved from magnetic tape.

The I option can be used in place of ID-SUPP.

Example:

The following example suppresses the printing of item IDs in a BOOKS report:

```
>SORT BOOKS BY TITLE TITLE PRICE ID-SUPP
```

```
PAGE   1                                          12:16:05  01 NOV 1989

TITLE.............................. PRICE...

DATABASE MANAGEMENT SYSTEMS             $9.95
OPERATING SYSTEM CONCEPTS              $18.75
WORD PROCESSING                        $22.98
WRITING COMMERCIAL APPLICATIONS        $24.50

4 ITEMS LISTED.

>
```

IF

Synonym for WITH. See WITH for description.

IN

IN is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>SORT THE ITEMS IN THE CUSTOMERS FILE BY
   LAST-NAME
>SORT CUSTOMERS BY LAST-NAME
```

The connectives THE, ITEMS, IN, and FILE are ignored in the first statement.

ITEMS

ITEMS is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>SORT THE ITEMS IN THE CUSTOMERS FILE BY
  LAST-NAME
>SORT CUSTOMERS BY LAST-NAME
```

The connectives THE, ITEMS, IN, and FILE are ignored in the first statement.

LE

LE is the relational operator LESS THAN OR EQUAL TO, used in selection expressions to compare data in an attribute to a constant.

Synonym: <=

Example:

The following example sorts orders with amounts less than or equal to "100":

```
>SORT ORDERS WITH TOTAL.AMT LE "100" TOTAL.AMT
```

*LIKE

Not implemented on all systems. ADDS Mentor uses the LIKE connective to select items with a data element that "sounds like" the specified constant. The LIKE connective converts an alphabetic text string into a phonetic equivalent based on a Soundex algorithm.

Prime INFORMATION and uniVerse use LIKE to find data that matches the specified constant. On those systems, LIKE is a synonym of MATCHING.

PTR

LPTR is an output modifier that sends a report to the line printer instead of to the screen.

The P option can be used in place of LPTR.

Example:

The following example sends a report from the ORDERS file to the line printer:

```
>SORT ORDERS BY DATE BREAK-ON DATE TOTAL.AMT
  LPTR
```

T

LT is the relational operator LESS THAN, used in selection expressions to compare data in an attribute to a constant.

Synonyms: <, BEFORE

Example:

The following example lists customers with a zip code less than "24767":

```
>LIST CUSTOMERS WITH ZIP LT "24767"
```

MATCHING

Not implemented on all systems. Prime INFORMATION and uniVerse use MATCHING to find data that matches a specified constant or string. See Chapter 3 for more information about string searching.

Synonyms: MATCHES, LIKE

N is an option that suppresses automatic paging on the terminal (NOPAGE).

NE

NE is the relational operator NOT EQUAL TO, used in selection expressions to compare data in an attribute to a constant.

Synonyms: #, NOT, NO

Example:

The following example selects customers who do not live in California:

```
>SELECT CUSTOMERS WITH STATE NE "CA"
```

NO

NO is a relational operator that selects items with a null value for a specified attribute.

Synonym: #, NOT, NE

Example:

The following example lists customers who have no data in the ZIP attribute:

```
>LIST CUSTOMERS WITH NO ZIP
```

NOPAGE

NOPAGE is a modifier that suppresses pagination so that a report scrolls continuously up the screen.

The N option can be used in place of NOPAGE.

Example:

The following example lists all customers and suppresses pagination:

```
>LIST CUSTOMERS NOPAGE
```

NOT

Synonym for "Not Equal To." See NE for description.

*NOT.MATCHING

Not implemented on all systems. Prime INFORMATION and uniVerse use NOT.MATCHING to find data that does *not* matches a specified constant or string. See Chapter 3 for more information about string searching.

O

O is an option used with the T-LOAD verb to overwrite an existing item.

OF

OF is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>SORT ORDERS BY THE DATE BREAK-ON THE DATE
  TOTAL OF THE AMOUNT
>SORT ORDERS BY DATE BREAK-ON DATE TOTAL
  AMOUNT
```

The THE and OF connectives are ignored in the first statement.

ONLY

ONLY is a file modifier that suppresses the default output specifications for a file and displays the item IDs only. It must immediately precede the filename in an ACCESS query.

Example:

The following example sorts the item IDs in the CUSTOMERS file dictionary:

```
>SORT ONLY DICT CUSTOMERS
```

OR

OR is a logical connective used to join multiple selection expressions. OR specifies that only one of the specified conditions must be met for an item to be selected.

The AND connective takes precedence over the OR connective when both appear in the same query. This can be altered by enclosing in parentheses the selection expression that is to be evaluated first.

OR is assumed when a logical connective is not explicitly entered between two selection expressions.

Example:

The following example lists all customers in Maine, New Hampshire and Vermont:

```
>LIST CUSTOMERS WITH STATE = "ME" OR STATE = "NH"
   OR STATE = "VT" STATE
```

P

P is an option that sends output to the printer (LPTR).

*SAID

Not implemented on all systems. Prime INFORMATION and uniVerse use SAID to select items with a data element that "sounds like" the specified constant. The SAID connective converts an alphabetic text string into a phonetic equivalent based on a Soundex algorithm.

SAID

Synonyms: SPOKEN, ~

*SPOKEN

Not implemented on all systems. Prime INFORMATION and uniVerse use SPOKEN to select items with a data element that "sounds like" the specified constant. The SPOKEN connective converts an alphabetic text string into a phonetic equivalent based on a Soundex algorithm.

Synonyms: SAID, ~

SUPP

Synonym for HDR-SUPP. See HDR-SUPP for description.

*T

T is an option that sends output to the terminal (the default).

TAPE

TAPE is a selection modifier that retrieves and displays items from a T-DUMP tape. TAPE can be used only with the LIST, LIST-LABEL, LIST-ITEM, SUM, STAT, ISTAT, HASH-TEST, and COUNT verbs.

Example:

The following example retrieves selected inventory items from magnetic tape:

```
>LIST ORDERS WITH DATE < "20 MAR 1986" TOTAL.AMT
  TAPE
```

THE

THE is a throwaway connective that can be used in an ACCESS query to make it sound more like an English-language sentence. It has no effect.

Example:

The following two statements produce the same report:

```
>SORT THE CUSTOMERS FILE BY THE LAST-NAME
>SORT CUSTOMERS BY LAST-NAME
```

The connectives THE and FILE are ignored in the first statement.

TOTAL

TOTAL is an output modifier that calculates and displays totals for numeric attributes. It must be placed immediately before the name of the attribute to be totalled.

The syntax for the TOTAL modifier is:

TOTAL *attribute* [*limiters*]

attribute	is the name of the attribute to be totalled.
limiters	totals only data matching the specified criterion. The criterion is expressed by a relational operator and a constant. The constant should be enclosed in double quotes or backslashes.

The total appears at the end of the report in the column of the specified attribute. It is preceded and followed by a blank line.

The total is identified by three asterisks in the item ID column. These can be replaced with any text by including the GRAND-TOTAL modifier in the ACCESS query.

When the TOTAL modifier is used with the BREAK-ON modifier, the report includes a subtotal for each control break as well as a total for the whole report.

Example:

The following example produces a report that totals the amount for all orders with an amount less than "100":

TOTAL

```
>LIST ORDERS WITH AMOUNT < "100" TOTAL AMOUNT
```

```
 PAGE   1                                          15:12:49  01 NOV 1989

ORDERS.... AMOUNT....

10116           $40.00
10119           $40.00
10122           $40.00
10101           $18.00
10123           $40.00
10102           $69.00
10114           $45.00
10106           $60.00
10121           $60.00
10124           $40.00

***            $452.00

10 ITEMS LISTED.

>
```

USING

USING is a connective that generates a report using the Attribute Definition items in a specified file rather than the data file's own dictionary.

The syntax for the USING modifier is:

USING [**DICT**] *filename*

Example:

The following example produces a report from the OLD-CUSTOMERS file using the Attribute Definition items in the CUSTOMERS dictionary:

```
>LIST OLD-CUSTOMERS USING DICT CUSTOMERS
```

*VERTICALLY

Not implemented on all systems. Prime INFORMATION and uniVerse use VERTICALLY to override the default columnar report format, listing each item's data in a linear format, with data from each attribute displayed on its own line. Multivalues are listed in columns underneath their column headings.

Synonym: VERT

*WINDOW

Not implemented on all systems. Ultimate uses the WINDOW modifier to begin a phrase that defines a window for printing multivalues on forms. WINDOW phrases are used in conjunction with forms generation expressions.

The syntax for the WINDOW modifier is:

WINDOW @(*x,y,z* [{ **1** | **2** }] **)** [**:** "*string* " **: @(***n* **)** **:** "*string* " ...]]

where *x*, *y*, and *z* specify the left-most column, top row, and bottom row of the window. "1" and "2" specify whether each multivalue should occupy one or two lines. A colon indicates concatenation. *n* specifies the column postiion on the current line where *string* is to be printed or displayed.

A WINDOW phrase must always be followed by the END-WINDOW modifier.

WITH

WITH is a selection modifier that introduces a selection expression.

The syntax for the WITH connective is:

WITH [**EACH**] *attribute-name* [[*rel-op*] *value-list*]
[{ **AND** | **OR** } **WITH** [**EACH**] *attribute-name* [[*rel-op*] *value-list*] ...]

EACH	specifies that all values in a multivalued attribute must meet the specified condition if the item is to be included in the report.
attribute-name	is the name of the attribute whose data values are to be compared to the specified condition.
rel-op	can be any relational operator.
value-list	can be either one or more data values, or a constant. Values should be enclosed in double quotes.
AND | OR	specifies a compound expression. AND requires that all conditions be met before an item is selected; OR requires that only one of the conditions be met.

Multiple WITH expressions can be used in an ACCESS query and are normally read left to right.

Synonym: IF

Example:

The following example lists all customers named Robert Smith:

```
>LIST CUSTOMERS WITH LAST-NAME "SMITH" AND WITH
   FIRST-NAME "ROBERT"
```

The next example lists all orders with either a part description of "SERIAL CABLE" or "CABLE - SERIAL":

```
>LIST ORDERS WITH DESC = "SERIAL CABLE" OR
   "CABLE - SERIAL"
```

WITHIN

WITHIN is a connective that retrieves *subitems* from a specified item.

A file is comprised of items containing data stored as attributes, values, or subvalues. In addition, subitems that provide a further description of the item can also be included in the file. Subitems are stored in exactly the same way as are regular items. To make a file item a subitem, its item ID is stored as a value in an attribute containing subitems. Multiple subitems can be made by storing item IDs as multivalues in the subvalue attribute.

Two conditions must be met in order for the WITHIN connective to work. First, the item specified by the WITHIN connective must contain a multivalued attribute whose values are item IDs (the subitems). Second, in the file dictionary the D-pointer to the data file must contain the following correlative in Attribute 8:

V;;*amc*

where *amc* is the number of the attribute containing the subitems. Both the item specified by WITHIN and the attribute containing the subitems must be in the same file.

The syntax for the WITHIN connective is:

WITHIN *filename* ' *item-ID* '

filename	is the name of the file.
item-ID	is an item that contains an attribute with subitems to be included in the report. Only one item ID can be specified per statement. Enclose the item ID in single quotes.

Each subitem encountered is assigned a *level number*. The item ID specified by the WITHIN connective is Level 1. If this item has subitems, they are assigned Level 2. If Level 2's subitems have subitems, they are assigned Level 3, etc., up to a maximum of 20 levels. Level numbers are printed in the report in front of the item and its subitems.

WITHIN

Example:

In the following example, the multivalued attribute SUB-PROD contains item IDs of the PRODUCT file. The following statement lists the item "A2000-1234" and three levels of subitems referenced by it:

```
>LIST WITHIN PRODUCT 'A2000-1234' PROD# DESC
  VALUE LOCATION SUB-PROD QOH ID-SUPP
```

```
E   1                                          15:03:25  01 NOV 1989

EL Prod #.... Description..... Value. Location.. Sub-Prod.. On Hand

   A2000-1234 SERVOS               0.73 R-123-8888 A2001-7811      73
                                                   A2001-8900
                                                   A2001-9112
   A2001-7811 D MOTOR           0.55 R-17-1001  A2002-1000     643
                                                   A2002-1023
   A2002-1000 D MOTOR PLATFORM  0.73 R-123-8888                  58
   A2002-1023 D MTR POWER UNIT  0.73 R-123-1002                  89
   A2001-8900 SERVO BOARD          0.12 L-44-1001                329
   A2001-9112 SERVO HOUSING        1.09 L-17-189   A2002-1032    107
                                                   A2002-1566
   A2002-1032 HOUSING SEALS        1.02 L-09-1889                768
   A2002-1566 HOUSING PLATES       1.03 L-1-3309   A2004-1111    355
   A2004-1111 HOUSING PACKAGE     12.00 R-12-1212                455

TEMS LISTED.
```

The item specified by the WITHIN connective is listed as LEVEL 1. LEVEL 2 lists subitems specified in the Level 1 item's SUB-PROD attribute. LEVEL 3 lists any subitems specified in the Level 2 item's SUB-PROD attribute. And one of the Level 3 items even contains a Level 4 subitem.

WITHOUT

Synonym for WITH NO. See WITH and NO for description.

*Y

Y is an option that displays translation of A-correlative DICT entries. This is useful as a debugging tool.

APPENDIX C

Correlative and Conversion Codes

Appendix C is a reference guide to all of the ACCESS correlative and conversion codes arranged in alphabetical order. Each entry comprises a brief explanation of what the code does, a complete explanation of its syntax, and a description of how to use the code. For examples showing how the codes work, see Chapter 8.

A CODE: Algebraic and String Functions

The A code is used to perform algebraic or string processing. It can be used to manipulate numeric or string data located in other attributes.

A [;] *expression*

An *expression* is one or more arithmetic or relational operators which operate on any of the following operands: *attributes* specified either by attribute number or by attribute name; *literals* in single or double quotes; special *system variables*; or *functions*. Evaluation of *expression* is from left to right; parentheses can be used to indicate the precedence of operations. All of these expressions are described in more detail in the following sections.

The A code is an easier-to-use version of the F code.

Attributes

An attribute can be identified by its attribute number or by its attribute name. When an attribute name is specified, correlatives and conversions, if present, are applied *before* the data element is used in *expression*. This may require that you mask decimals with scaling (see the ML and MR codes).

attribute#[R [R]]	the attribute number that refers to the position of data in the file. An optional R code can be used with multivalued attributes to specify that the first value of an attribute is to be used repeatedly in an operation involving other multivalued attributes. A second R specifies that the first subvalue is to be used repeatedly with other multisubvalued attributes.
N(*name*)	an attribute name as defined in the dictionary. If the name is not found, an error message is returned.
9998	is the current item count.
9999	is the current item size.

Literals

A literal, either a string or a numeric constant, must be enclosed in single or double quotes. A numeric value can be any positive or negative integer, or zero.

System Variables

The following system variables are available as operands in any expression:

D	indicates the system date in internal format.

LPV means **l**oad **p**revious **v**alue; that is, load the result of the last correlative or conversion for further processing.

NB is a break level counter, incremented with each break that is encountered. The lowest-level break is 1; the break that specifies the grand total is 255.

ND is a detail line counter. This supplies the total number of detail lines at each breakpoint.

NI is an item counter, incremented for each item in an output list.

NS is a subvalue counter, incremented for each subvalue of an attribute.

NV is a value counter, incremented for each value in a multivalued attribute.

T indicates the system time in internal format.

Functions

R(*operand1*,*operand2*)
: is the remainder after dividing the first operand by the second operand. An operand can be any valid expression.

S(*expression*)
: sums all multivalues in *expression.*

attribute[*start*,*length*]
: extracts a substring. *attribute* can be identified by attribute number or by attribute name. Attribute numbers, literals in single or double quotes, or expressions can be specified within the brackets. Brackets are part of the syntax here and must be typed. *start* is the starting position of the first character to be extracted, *length* is the number of characters to be extracted.

IF *expression1* THEN *expression2* ELSE *expression3*
: is a conditional statement that evaluates *expression1* and, if true, then evaluates *expression2*, or if false, evaluates *expression3*. This function is not available on all systems, and is not included in the SMA standards.

Arithmetic Operators

+	Addition.
–	Subtraction.
*	Multiplication.
/	Division (returns an integer).
:	Concatenation.

Relational Operators

=	Equal to.
#	Not equal to.
>	Greater than.
<	Less than.
>=	Greater than or equal to.
<=	Less than or equal to.

Precedence of Operations

The precedence of operations is as follows:

1. Multiplication and division.
2. Addition and subtraction.
3. Relational operations.

If two operators have the same precedence, they are evaluated from left to right. Use parentheses to specify the order for evaluating each operation; up to 20 nested levels of parentheses are permitted.

C CODE: Concatenation

The C code is used to concatenate attributes or literals, or both.

C *expression1 c expression2* [*c expression3...*]

c	is a character that separates concatenated elements. Any nonnumeric character except a semicolon or a system delimiter is a valid separator, including a blank. A semicolon (;) indicates that no separator is to be used.
expression	specifies each element to be concatenated. An *expression* can be either an attribute number; any literal string enclosed in single or double quotes or backslashes; or an asterisk. An asterisk specifies the result generated by a previous conversion or correlative operation.

Attributes in expressions can be specified only by attribute number; they cannot be specified by name.

Any number of expressions and separation characters can be included in any order.

When the Attribute Definition item containing the C code is set up in the file dictionary, the attribute number (line 2) should be zero. If any other attribute number is used and the attribute specified contains a null value, the concatenation will not be performed.

D CODE: Date Conversion

The D code converts dates to internal format and, upon output, to one of several different external formats.

D CODE

D [*year*] [{ *separator* | *subcode* }]

year is a digit from 0 through 4 that specifies the number of digits used to represent the year. Zero signifies that no year is output; 4 is the default.

separator is any single nonnumeric character that is used to separate the day, month, and year (if specified) on output. Usually this separator is a slash (/) or a hyphen (-).

subcode can be any of the following special date subcodes, if no separator is specified:

D	lists only the number of the day.
I	lists date in internal format (reverse conversion).
J	lists the Julian day of the year.
M	lists only the number of the month.
MA	lists only the name of the month.
Q	lists only the number of the quarter.
W	lists only the number of the day of the week (Sunday is 7).
WA	lists only the name of the day of the week.
Y	lists only the number of the year.

If *subcode* is not specified, the date format will be 12 DEC 1967; if *subcode* is specified, the format will be 12/12/1967.

Dates are almost always stored in a special internal format that uses less disk space than conventional date formats and makes it easier to perform arithmetic operations on this data. The internal representation of a date is the number of days before or after the zero date, which is December 31, 1967.

Date conversions are almost always located in Attribute 7 of an Attribute Definition item because it is faster for the system to convert an external date to an internally stored format once and then compare the stored date to data contained in other items of the file. It is possible, however, to specify a date conversion in Attribute 8. This causes slower processing,

because the internal date of every item searched must be converted to external format. This is necessary, for example, when the selection criteria or sort keys do not contain the day, month, and year.

If only two digits of the year are specified, year 30 to 99 are stored as 1930 to 1999, and 00 to 29 are stored as 2000 to 2029.

Doing a Group Extraction with the D Code

You can also specify in the syntax line that the D code first do a group extraction to obtain the stored date. The syntax line in this case is:

D [*year*] [*cm*] [*subcode*]

where *c* is any single nonnumeric character used as a delimiter to separate fields (it cannot be a system delimiter), and where *m* is a single digit that specifies the number of fields to skip in order to extract the date. *m* must be specified if *c* is specified.

In the following examples, the data is stored as "ABC*8602".

If conversion is...	Then output is...
D2*1	ABC*20 JUL 91
D2*1/	ABC*07/20/91

F CODE: Stack Functions

The F code performs mathematical and string processing operations on specified attributes or constants. It is made up of operands and operators in reverse Polish format separated by semicolons.

There are two versions of the F code: F and FS. F is the older version, FS (S is for "standard") is the newer SMA standard version. The main difference between the two is in the handling of relational operators.

F [;] *element* [; *element* ;...]
FS [;] *element* [; *element* ;...]

The program processes the F code from left to right, putting each element on the stack as an operand until it encounters an operator.

The semicolon is used to separate elements. An *element* can be any of the following:

- an attribute number.
- a literal.
- a system variable.
- a function.
- an arithmetic operator.
- a relational operator.

Each of these is described in the following sections.

Attributes

An attribute can be referenced only by its attribute number.

attribute#[R [R]] is the attribute number that refers to the position of data in the file. An optional R code can be used with multivalued attributes to specify that the first value of an attribute is to be used repeatedly in an operation involving other multivalued attributes. A second R specifies that the first subvalue is to be used repeatedly with other multisubvalued attributes.

Literals

Any literal string must either be enclosed in single or double quotes or be preceded by a C, thus:

C*n*

where *n* is a numeric constant to be pushed onto the stack. If C is used, the literal string ends at the next semicolon.

System Variables

The following system variables are available as operands in any expression:

D indicates the system date in internal format.

LPV means **l**oad **p**revious **v**alue; that is, load the result of the last correlative or conversion for further processing.

NB is a break level counter, incremented with each break that is encountered. The lowest-level break is 1; the break that specifies the grand total is 255.

ND is a detail line counter. This supplies the total number of detail lines for each breakpoint.

NI is an item counter, incremented for each item in an output list.

NS is a subvalue counter, incremented for each subvalue of an attribute.

NV is a value counter, incremented for each value in a multivalued attribute.

T indicates the system time in internal format.

Functions

_ (underscore) exchanges the first and second elements in the stack.

: concatenates the first element in the stack to the end of the second element.

P duplicates the first element and pushes it onto the stack.

R divides the first element in the stack by the second element, putting the remainder on top.

S sums all values in the first element of stack 1.

[ ] extracts a substring. This function uses the top three elements in the stack, where:

Top	*length*
2	*start*
3	*string*

The third element on the stack is the string to be operated on. The second element is the character position at which extraction starts. The first element is the number of characters to extract. The result is placed on top of the stack.

(*conversion code*)
applies a conversion on the first element in the stack.

Arithmetic Operators

The following arithmetic operators work on the top two elements in the stack. The operands are popped and the result is pushed on to the stack as the first element. Arithmetic operations are carried out on integers only; any decimal places are disregarded. Nonnumerics are treated as zeros and no error message is displayed.

\+ adds the first and second elements in the stack.

– subtracts the first element from the second element.

*[*n*] multiplies the first and second elements. The optional *n* is the descaling factor, that is, the result is divided by 10 raised to the *n*th power.

/ divides the second element by the first element.

Relational Operators

The following operators work on the top two elements in the stack and return a result of one or zero to the top of the stack, depending on whether the condition is or is not satisfied:

=	Equal to.
#	Not equal to.
>	Greater than.
<	Less than.
]	Greater than or equal to.
[	Less than or equal to.

G CODE: Group Extraction

The G code extracts one or more fields from a delimited character string.

G [*skip*] *delimiter #fields*

skip	specifies the number of fields to skip. If *m* is not specified, zero is assumed and no fields are skipped.
delimiter	is the field separator. This delimiter can be any single nonnumeric character except a minus sign (–) or a system delimiter (SM, AM, VM, SVM, or SB).
#fields	is a decimal number indicating the number of contiguous delimited fields to extract. If *n* = 0, a null value is returned.

L CODE: Length Validation

The L code tests the length of data to determine whether or not it should be output. The L code without any parameters returns the length of the data element.

L CODE

L [*n* [,*m*]]

n	specifies a maximum length of *n* characters. Data exceeding this length produces a null value. If *n* = 0, the length of the value is returned.
n,m	specifies a range of *n* to *m* characters. Data that does not fall within this range produces a null value.

MC CODES: Character Masks

The Masked Character conversion codes allow the user to translate data from one case to another or to extract certain classes of characters.

MCA	extracts all alphabetic characters, both upper- and lowercase. Nonalphabetic characters will not be printed.
MC/A	extracts all nonalphabetic characters. Alphabetic characters will not be printed.
MCD[X]	converts numeric data from decimal format to its hexadecimal equivalent.
MCL	converts all uppercase letters to lowercase. Does not affect lowercase letters or nonalphabetic characters.
MCN	extracts all numeric characters. Alphabetic characters will not be printed.
MC/N	extracts all nonnumeric characters. Numeric characters will not be printed.
MCP	converts nonprintable characters to dots and drops the high-order bit of all characters above character 127.
MCT	converts the first character of each word to uppercase and all other characters to lowercase. Does not affect nonalphabetic characters. A word

begins after any nonalphabetic character except a single or double quote.

MCU converts all lowercase letters to uppercase. Does not affect uppercase letters or nonalphabetic characters.

MCD[D] converts numeric data from hexadecimal format to its decimal equivalent.

Using MC Codes

If input is...	and conversion is...	then output is...
BLEAR34TRE	MCA	BLEARTRE
BLEAR34TRE	MC/A	34
BLEAR34TRE	MCL	blear34tre
#$abc123	MCN	123
#$abc123	MC/N	#$abc
#$abc123	MCU	#$ABC123
JOHN SMITH	MCT	John Smith
567	MCX	1383
FB	MCX	251
33D	MCX	829
5332	MCD	14D4
111	MCD	6F
4444	MCD	115C

ML and MR CODES: Formatting and Scaling Numbers

The ML and MR codes allow special processing and formatting of numbers and dollar amounts. ML specifies left justification, MR specifies right justification.

M { **L** | **R** } [*n* [*m*]] [**Z**] [,] [*options*] [(*format-mask*)]

n specifies the number of digits to be printed to the right of the decimal point. If *n* is omitted or is zero, no decimal point will be printed.

m specifies that the number is to be divided by that power of 10. If not specified, *m* = *n* is assumed. If *m* is greater than *n*, the number is rounded off to *n* digits.

Z indicates that a data value of zero is to be output as null.

, specifies that commas are to be inserted every three digits to the left of the decimal point.

options can be any of the following:

$ places a floating dollar sign in front of the number.

C causes negative values to be followed by the letters CR.

D causes positive values to be followed by the letters DB.

E causes negative numbers to be enclosed in angle brackets (< >).

M causes negative numbers to be followed by a minus sign.

N suppresses the minus sign on negative numbers.

If no letter options are specified, negative numbers are listed with a leading minus sign.

format-mask can be one of the following three codes:

#*n* specifies that data is to be justified in a field of *n* blanks.

**n* specifies that data is to be justified in a field of *n* asterisks. Any fill character can be used in place of the asterisk.

%*n* specifies that data is to be justified in a field of *n* zeros.

Literal strings can also be enclosed in parentheses. The text will be printed as specified, with the number being processed right- or left-justified.

MT CODE: Time Conversion

The MT code converts time to internal format and, on output, to 12-hour or 24-hour format. If no options are specified, time is represented in 24-hour format (*hh* : *mm*).

MT [H] [S]

H specifies 12-hour format with AM or PM appended. If H is omitted, 24-hour format is assumed.

S specifies that seconds are to be included. If S is omitted, seconds are not listed.

Data representing time is usually stored internally as the number of seconds after midnight.

☞ The MT code can give unusual results on numbers greater than 86399, since only those numbers in the range from 0 (12:00 PM) through 86399 (11:59 PM) are valid for time conversions.

MX and MY CODES: Character-to-Hexadecimal Format

The MX code converts a character string to its hexadecimal ASCII equivalent. The MY code converts a hexadecimal character string to its alphanumeric equivalent

MX
MY

The MX code converts each character to a two-byte hexadecimal number. This conversion is useful for finding nonprintable characters in strings of data.

The MY code converts each two-byte hexadecimal number to a single ASCII character.

CODE: Pattern Matching

The P code validates data if it matches the specified pattern. If the data does not match the specified pattern, a null value is returned.

P (*pattern*) [; (*pattern*)...]

pattern can contain any of the following codes:

*n*A	is an integer followed by the letter A, which tests for *n* alphabetic characters.
n N	is an integer followed by the letter N, which tests for *n* numeric characters.
*n*X	is an integer followed by the letter X, which tests for *n* alphanumeric characters.
'*xxxx*'	is a literal string which must match the same literal string in the data. A literal string must be enclosed in single quotes. Single-character delimiters such as hyphens or slashes need not be enclosed in quotes.

If *n* is 0, any number of numeric, alphabetic, or alphanumeric characters will match.

Each pattern to be matched must be enclosed in parentheses.

CODE: Range Validation

The R code limits returned data to that which falls within specified ranges.

R *n* , *m* [; *n* , *m* ;...]

n is the lower bound.

m is the upper bound.

If you specify a single range, the data returned is that which falls in the range between *n* and *m* (inclusive). If you specify multiple ranges, the data that falls in any of the ranges is returned. State these ranges in ascending numerical order: for example, R3,5;9,14;16,30. This returns data that falls within the ranges 3-5, 9-14, or 16-30.

If you specify a range to search for negative numbers, you must state the negative number first.

If range specifications are not met, a null is returned.

S CODE: Substitution

The S code replaces the value of the referenced attribute (that is, the attribute referenced in line 2 of the Attribute Definition item) with a value from another attribute or with a literal string.

S ; *element1* ; *element2*

element# can be either an attribute number or a literal string. Literal strings must be enclosed in single quotes. If the value in the referenced attribute is *not* null or zero, then *element1* is substituted for the value. If the value in the referenced attribute *is* null or zero, then *element2* is substituted for it.

☞ On ADDS systems the value in the referenced attribute must be zero in order for the S code to work; if the value is null, the S code has no effect.

T CODE: Text Extraction

The T code extracts a specified number of characters from a string beginning at a specified position.

T [*start* ,] *#chars*

start is the leftmost position of the first character to extract. If *start* is omitted, 1 is assumed.

#chars is the number of characters to extract.

If *start* is specified, characters will be extracted from left to right. If *start* is not specified, the direction is determined by the justification specified in line 9 in the Attribute Definition item.

T*file* CODE: File Translation

The T*file* code translates data from one file to another file. It uses data stored in one file to reference the item IDs of another file in order to retrieve data from the second file. This eliminates the need for duplicating data in related files.

T [DICT] *filename* **;** *code* **[** *vmc* **] ;** *in-attr#* **;** *out-attr#* **[;** *br-attr#* **]**

DICT specifies the file dictionary from which data is to be translated.

filename is the name of the file from which data is to be translated.

code is an action code, which specifies the action to be taken if the item being looked up does not exist. This code must be one of the following:

- C If conversion is impossible, return the original data.
- I Input verify only. Functions like V for input and like C for output.
- O Output verify only. Functions like C for input and like V for output.
- V Conversion item must exist on file, and the specified attribute must contain data, otherwise an error message is returned.
- X If conversion is impossible, return a null value.

vmc is a value mark count indicating the value to be returned from a multivalued attribute. If *vmc* is

not specified and the attribute is multivalued, all values are returned with all system delimiters turned into blanks.

in-attr# is the number of the attribute for *input* conversion. *in-attr#* is used only in Pick BASIC using ICONV. Although *in-attr#* is not used by ACCESS, it cannot be left out of the T*file* code. It can be either null or the same as the *out-attr*.

out-attr# is the number of the attribute for *output* translation. The data to be translated is retrieved from the attribute specified by the *out-attr#*.

br-attr# is the number of the attribute that is used instead of *out-attr#* during the listing at BREAK-ON and TOTAL time if the translation is in Attribute 7 of the Attribute Definition item.

The T*file* code in the *source* file translates data from the *target* file. The source file must include an attribute whose data is the same as the item IDs of the target file. This attribute in the source file contains the keys—the item IDs of the target file—that allow the source file to access any of the data in the target file. The T*file* code specifies (1) the *name* of the target file, (2) the *attribute number* of the attribute in the target file from which data is to be translated, and (3) what *action* is to be taken if there is no data value in that attribute.

APPENDIX D

File Dictionary Structures

Appendix D describes the structure of the File Definition items and Attribute Definition items that are contained in file dictionaries.

A Pick system database file comprises two physical files: a dictionary that describes the structure of the data and a data file that contains the actual data stored on disk. In other words, the raw data is kept in one place and the definition of that data is stored separately in the file dictionary. A piece of data can be stored in one place, unencumbered by its description. The dictionary contains this description; it might contain multiple descriptions of the same piece of data. This arrangement provides great flexibility in setting up a database, permitting different logical views of the same set of data.

A data file is organized as a collection of variable-length items. An item is organized as a sequence of variable-length *attributes*. Each item in the file is identified by a unique item ID.

Each data file is associated with a file dictionary. The dictionary points to the data file and defines its structure. Several data files can share one file dictionary if their data structure is similar.

Some dictionaries do not have an associated data file. These dictionaries are called single-level files and might contain data rather than dictionary entries. A single-level file points to itself. It can be referenced either as a dictionary or as a data file.

Dictionaries contain items just as data files do. Dictionary items are sometimes called *descriptors* because they describe the data stored in the data

file. The two main types of descriptor are: File Definition items and Attribute Definition items. File Definition items are known as D-pointers because they point to the data file itself. Attribute Definition items either describe data stored in a particular attribute, or they derive data from attributes using processing codes (correlatives). Each of these descriptor types is summarized in the following sections.

File Definition Items

File Definition items are D-pointers that point to the location of a file (any file). The file can be a single-level file, in which case it has no data-level file associated with it. Most files, however, have two levels. In this case, the File Definition item in the Master Dictionary points to a file dictionary which, in turn, contains a File Definition item that points to a data file.

Item ID. A File Definition item is a pointer to a dictionary or data file. The item ID is the actual name of the referenced dictionary or data file. If it points to a data file, then the item ID is the same as the name of the file.

Attributes. File Definition items can have 13 attributes. Attributes 4, 11, and 12 are reserved and are therefore empty.

1 **Definition Code.** A "D" indicates a File Definition item. Other file definition codes are also used. "DC" is used for files whose dictionaries contain items that point to frames containing BASIC object code and whose data files contain the source code. DC files are also used for storing select-lists.

2 **Base.** The base frame ID indicates the starting location of the file. This value is assigned automatically by the CREATE-FILE processor. Do not change this value.

3 **Modulo.** The modulo indicates the number of contiguous groups originally occupied by the file. It is supplied as a parameter by the user during file creation. Do not change the modulo in Attribute 3.

4 **Separation.** The separation indicates the number of contiguous frames that are to make up a group. On some systems it is supplied as a parameter by the user during file

creation. (Systems with larger frame sizes may no longer use this parameter.) Do not change the separation in Attribute 4.

5 **Retrieval Code.** A security code that restricts access to a file. When specified, its value must match any retrieval code in the Account Definition item for the file to be accessed.

6 **Update Code.** A security code prevents modification of a file. When specified, its value must match any update code in the Account Definition item for the file to be modified.

7 **Conversion Code.** A conversion specification that will be applied to the item ID.

8 Reserved.

9 **Justification.** A code that specifies the justification of the item ID in a column. "L" and "U" indicate left-justification and "R" indicates right-justification.

10 **Width.** A positive integer indicating the maximum column width of the item ID column. The SMA standard default for column width is 9.

11 Reserved.

12 Reserved

13 **Reallocation.** One or two numbers that specify a new modulo, or modulo and separation, for the file. The parameters will be used to reallocate the file during a file-restore.

Attribute Definition Items

Attribute Definition items define the format of items in the data file. These items can be created using the Editor.

Item ID. A logical name used to reference the contents of the attribute in ACCESS statements.

Attributes. Attribute Definition items can have 10 attributes. Attributes 5 and 6 are reserved and are therefore empty.

1 **Definition Code.** Any one of these Definition Codes specify an Attribute Definition item:

A An Attribute Definition item that references the contents of an attribute in the data file.

S An Attribute Synonym. Same as "A" for all practical purposes. (The LISTDICT Proc sorts S after A codes). Attribute synonyms are usually created as a numerical sequence of item IDs to define the default output specifications for ACCESS reports.

I An ACCESS phrase follows in line 2. Not used on all systems.

X A placeholder item that is skipped when maintaining a sequence of numeric item IDs to specify the default output specification for ACCESS reports. Data in the attribute referenced by an X item is not displayed.

2 **Attribute Number.** An integer identifying the attribute by its sequential location. Also called the Attribute Mark Count (AMC). An attribute number of 0 is used to reference the item ID. A 0 or any number greater than the number of attributes that exist in the file can be used to reference data that is derived or computed rather than actually stored on disk. In addition, if line 1 defines a phrase, the phrase itself starts on line 2.

3 **Column Heading.** An optional name that is used in place of the item-ID as a column heading in ACCESS reports. The backslash (\ ) is a reserved symbol causing no column heading to be output. You can use value marks (CTRL-]) to indicate a line break, making it possible to place a column heading on multiple lines. If the heading is shorter than the column width, it is padded with dots (...).

4 **Structure.** An optional code that defines an associative structure for two or more multivalued attributes. A structure code of "C" indicates a Controlling attribute; a structure code of "D" indicates a Dependent attribute. Dependent attributes can be listed in an ACCESS report only when the Controlling attribute appears in the output specification.

5 Reserved.

6 Reserved.

7 **Conversion Code.** A conversion specification that is to be applied to the contents of the attribute.

8 **Correlative Code.** A correlative specification that is to be applied to the contents of the attribute.

9 **Justification.** A code that specifies the justification of the output field in a column. It also affects the sorting of data.

- L Left-justified (generally the case for alphabetic data). If data exceeds the length defined by the maximum column width, it wraps to the next line, beginning at the first character over the maximum length. This is the default.
- R Right-justified (generally the case for numeric data).
- T Left-justified. This type of justification is used for textual data that might include embedded blank spaces. If data exceeds the length defined as the maximum column width (line 10), it wraps to the next line, starting at a blank space.
- U Left-justified. If data exceeds the length defined by the maximum column width (line 10), it does not wrap, but overwrites any data in the adjoining column.

10 **Column Width.** A positive integer indicating the maximum column width of the column.

Nonstandard File Dictionaries

Some Pick systems use nonstandard file dictionaries. These dictionaries perform most of the same functions as the Pick/SMA standard dictionaries do, but their structures are different. Prime INFORMATION and uniVerse systems both use a different dictionary structure, although they both support the Pick/SMA standard dictionary format as an alternative. The following sections describe the make-up of these dictionaries.

Prime INFORMATION Dictionaries

Prime INFORMATION and uniVerse dictionaries contain the following types of entry:

- Data descriptors, which define the format and location of attributes in each item of the data file. These perform functions similar to the standard Pick Attribute Definition items.
- I-descriptors, which define the format and location of an interpretive attribute whose value is derived from other attributes in the data file or from attributes located in other data files. They are called I-descriptors because the attributes defined by them are interpretive. I-descriptors perform functions similar to those of Pick Attribute Definition items containing correlative codes.
- @-phrases, which contain user-defined default output specifications.
- User-defined phrases that contain part of an ACCESS sentence (such as selection expressions, sort expressions, output specifications, etc.—anything, in fact, except an ACCESS verb).
- X-descriptors, which contain other user-defined information. These are available only on uniVerse systems.

Data Descriptors and I-Descriptors

Data descriptors define the format of items in the data file. These items can be created using either the Editor or a data-entry utility such as REVISE (uniVerse) or ENTROC (Prime INFORMATION).

Field Name. A logical name used to reference the contents of the data descriptor in ACCESS statements. Equivalent to item ID.

Fields. Data descriptors normally have seven fields (attributes), as follows:

1 **Type and Description.** A "D" in field 1 defines a data descriptor, an "I" defines an I-descriptor.

2 **Location.** Equivalent to the attribute number (AMC). This number identifies the field by its sequential location. In I-descriptors, this field contains the expression to be evaluated;

thus Field 2 in I-descriptors is similar to the correlative attribute (line 8) in Pick Attribute Definition items. I-descriptor expressions in Field 2 are not limited to correlative codes, however, but can also include a wide range of subroutines and predefined functions.

3 **Conversion.** Equivalent to the conversion attribute (line 7) in Pick Attribute Definition items. Field 3 contains a conversion specification that is to be applied to the contents of the field.

4 **Column Heading.** Equivalent to the column heading attribute (line 3) in Pick Attribute Definition items. An optional display name that is used in place of the item ID as a column heading in ACCESS reports. If it is not specified, the field name is displayed.

5 **Output Format.** Equivalent to the justification and column width attributes (lines 9 and 10) in Pick Attribute Definition items. This field contains first a number indicating the maximum width of the display column used in ACCESS reports, then a letter that specifies the justification of the output field in the column. Justification can be "L" (left), "T" (left text), or "R" (right).

6 **Single/Multivalue.** An "S" in Field 6 defines the field as single-valued; an "M" defines it as multivalued.

7 **Association.** Contains the name of a phrase that relates two or more multivalued fields together such that for every value in one field there is an associated value in the other fields.

@ID Descriptors

Prime INFORMATION and uniVerse dictionaries do not contain D-pointers to the data file. Instead, they have an @ID descriptor that functions in similar ways to define the item ID "field" of the data file. When a file is first created, the only item the file dictionary contains is an @ID item, which is created automatically. It contains the following default structure, all fields of which can be modified except for field 2, Location, which must always be zero, and Field 6, which must always be "S".

@ID. The item ID of the descriptor.

Fields. @ID descriptors, like data and I-descriptors, also have seven fields (attributes), as follows:

1 **Type and Description.** Contains a "D" and the default description.

2 **Location.** Contains a zero, which defines the location of the item ID "field."

3 **Conversion.** No conversion specified.

4 **Column Heading.** Contains the name of the file, which is used as a default column heading for the item ID column.

5 **Output Format.** Contains either 10L or 12L, depending on the file type. Thus, column widths for the item IDs are either 10 or 12 characters, and the column is left-justified.

6 **Single/Multivalue.** Contains an "S" to define a single-value field. the item ID "field" must be single-valued.

7 **Association.** The user can supply the name of a phrase that relates two or more multivalued fields together such that for every value in one field there is an associated value in other fields.

INDEX

SYMBOLS

A

D

E

N

O

P

Q

R

S

V

W

Y

Z

Pick ACCESS

A GUIDE TO THE SMA/RETRIEVAL LANGUAGE

Until now, you've had to choose between reading Pick books that are understandable but shallow, and plowing through a difficult documentation set. No more. *Pick ACCESS: A Guide to the SMA/RETRIEVAL Language*, is the only book on Pick ACCESS you'll ever need. *Pick ACCESS* introduces ACCESS concepts; documents all commands, features, and functions; and includes a thorough description of correlatives and conversions.

The book includes chapters on:

- An overview of ACCESS
- ACCESS syntax
- Using selection and sort expressions
- Formatting reports
- Using select-lists
- Specialized processing
- Forms generation
- Correlatives and conversions

as well as an index and several helpful appendixes.

The goal of the Pick Series is to provide Pick documentation that is user-oriented: to help new users learn about Pick quickly and to help experienced users find accurate information easily. The Pick Series tackles almost all of the Pick system at a level of depth not found elsewhere, even in the original Pick manuals. It offers a complete Pick documentation set for all users, based on a mature implementation of the Pick operating system (R83), with notes on SMA standards and specific differences among major Pick implementations.

366 pages

ISBN 0-937175-41-2

O'Reilly & Associates, Inc.